UNIVERSIDAD CARLOS III

Escuela Politécnica Superior

Departamento Ingeniería de Sistemas y Automática

ANÁLISIS DE LOS ACCIONADORES DEL ROBOT HUMANOIDE RH-2

PROYECTO FIN DE CARRERA

INGENIERÍA TÉCNICA INDUSTRIAL
ELECTRÓNICA INDUSTRIAL

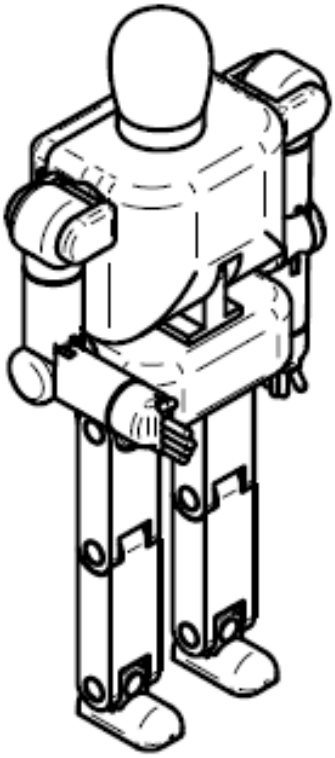
Autor: Alberto Navarro Criado
Director: Carlos Pérez Martínez

Junio 2009

Capítulo 4: EL CANBUS

ÍNDICE DE FIGURAS:

ÍNDICE DE TABLAS:



Capítulo 1: Introducción

Breve presentación del proyecto,
de los objetivos marcados
y descripción del libro.

1.1 INTRODUCCIÓN

En el departamento de Ingeniería de Sistemas y Automática de la Universidad Carlos III de Madrid, se están realizando diversos estudios e investigaciones dentro del campo de la robótica. El presente trabajo pretende mostrar el estado actual del sistema de control del movimiento de un robot humanoide autónomo que forma parte del proyecto RH-2 que se realiza en este departamento. El objetivo general del proyecto es desarrollar el tercer robot de la serie RH.

Las razones para seguir desarrollando los robots humanoides son evidentes. Éstos han de adaptarse cada vez mejor a los ambientes cotidianos diseñados para el hombre. Ambientes con escaleras, puertas, obstáculos, etc. Con esta necesidad, y con miras a mejorar la calidad de vida, se desarrollan este tipo de robots. Tener un asistente ideal, que realice también operaciones peligrosas o dificultosas para los humanos.

El robot tiene que ser capaz de realizar diversas tareas tanto en el sector industrial como en el de servicios, así como trabajar por sí solo e interaccionar con los seres humanos. El humanoide debe ser lo más parecido al hombre, por ello, sus características fundamentales son:

- Locomoción bípeda.
- Posesión de dos brazos para manipular objetos.
- Un cuerpo donde ubicar todos los elementos de control.
- Una cabeza provista de sensores de visión y de sonido mediante los cuales el robot pueda orientarse y navegar en su entorno de trabajo, así como recibir comandos por voz.
- Completa autonomía en cuanto a su capacidad energética y de toma de decisiones.
- Desempeño de sus funciones en entornos estructurados y que abarquen un amplio espectro de posibilidades, desde naves industriales hasta viviendas, oficinas, locales comerciales, escuelas, hospitales, ambientes exteriores, etc. Por ello el robot deberá ser capaz de andar pos distintos tipos de superficies, ya sean planas, inclinadas e incluso deberá poder subir y bajar escaleras.
- Las tareas a realizar por el robot son básicamente de tres tipos:
    - manipulación y transporte de objetos, tanto individualmente como en cooperación con otros robots o humanos.
    - realización de operaciones de montaje, mecánicas o similares.
    - desarrollo de funciones de carácter visual como inspecciones o vigilancia.

Hay que tener en cuenta, sin embargo, las limitaciones que existen actualmente y que son difícilmente solventables. En primer lugar, en la simulación del movimiento humano, nos encontramos con limitaciones físicas ya que para poder desplazarse el robot necesita un control y coordinación de todos los grados de libertad. Por otro lado tenemos una gran problemática en cuanto a lo que en baterías se refiere. Nos encontramos que llenan un gran espacio del disponible en el cubículo del robot y que se acumula un gran porcentaje de peso en ellas. Pese a esto, no nos aportan la suficiente autonomía energética como para permitir al robot desarrollar tareas de prolongada duración. La toma de decisiones será llevada a cabo por ordenadores que se encargarán de controlar el movimiento de las articulaciones, de controlar la adquisición de la información que procede de los sensores, coordinarlo con los actuadores y la comunicación con el exterior.

Por lo que se puede observar, este proyecto es extremadamente complejo y por ello se opta por repartir el trabajo entre un grupo del departamento. En este proyecto de fin de carrera solo se estudiará el funcionamiento de los controadores de los motores.

1.2 OBJETIVO

El objetivo fundamental de este proyecto es desarrollar una primera base sobre la que se edifique el sistema de control de todos los grados de libertad del robot RH-2 que permita el desempeño de las funciones para los cuales el robot ha sido creado. El estudio se va a centrar en los actuadores que transmiten las órdenes de la unidad de control a los motores, comprobando cuál de los disponibles en el mercado es ideal para nuestro proyecto. El controlador va a ser estudiado para generar un manual de usuario que pueda ser usado por otros miembros del departamento. Para conseguir este resultado se pueden definir los siguientes pasos a seguir:

- Análisis del estado del arte y establecimiento de las especificaciones generales del sistema de control a desarrollar, partiendo del actuador usado en el prototipo anterior desarrollado por el departamento.
- Análisis del estado actual del mercado, seleccionando posibles candidatos comprobando que éstos cumplen las especificaciones generadas en el apartado anterior.
- Adquisición del actuador surgido del apartado anterior y realización de pruebas para comprobar que cumple los requisitos establecidos.
- Generación de la documentación.

Como es de suponer, cuanto más se analicen las características del controlador así como todas las funciones que posea, mayor beneficio se obtendrá pues puede que algún elemento sea clave para el desarrollo del prototipo en la actualidad o a medio plazo, cuando se busque optimizar el sistema de control una vez desarrollado.

1.3 MOTIVACIONES

El ser humano es considerado una máquina perfecta y los ordenadores como el mayor invento de la humanidad. El uso de microprocesadores para construir un producto semejante a un humano supone el mayor reto para el mundo tecnológico. El desarrollo de un humanoide requiere recursos de todas las ramas de la ingeniería, en especial electrónica y mecánica.

Japón se ha considerado a partir de los 80 el precursor de nuevos productos y nuevos tecnologías. En este caso está mucho más avanzado que el resto del mundo en el desarrollo de humanoides. Se trata de un mercado poco rentable debido a los altos costes de los robots y sistemas aún en prueba. Pero teniendo en cuenta el avance de la tecnología y la rapidez con la que bajan los costes, puede convertirse Japón en el foco de este nuevo mercado. Ya pasó durante el siglo pasado con la creación de robots industriales. Las mayores multinacionales (Mitsubishi) que comercializan robots para industrias pertenecen a los países asiáticos junto con la europea ABB. Dos décadas fueron suficientes para la expansión del mercado a todo el mundo de la producción.

Es imprescindible que Europa investigue en este campo de la robótica para no quedar en desventaja respecto a los competidores. La mayoría de los proyectos europeos acerca de robots humanoides, no son comparables en cuanto al tamaño de inversión y calidad de los japoneses. EEUU está incluso menos actualizada que Europa y sólo destacan las investigaciones de la NASA en el campo de robots humanoides (ver figura en anexos).

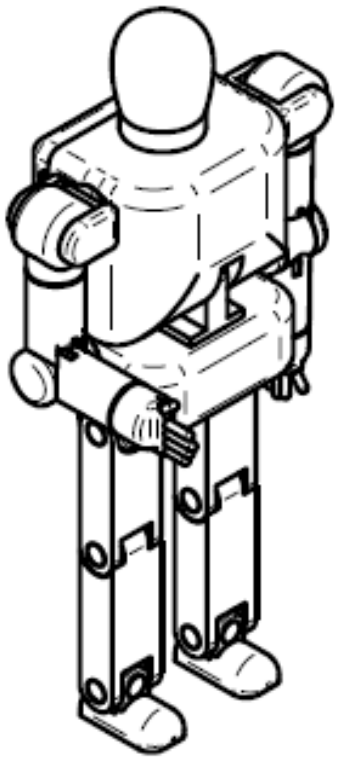
Por tanto es obvia la motivación para conocer los métodos de control de humanoides y el diseño de los sistemas hardware y software que necesita. Desde el primer proyecto del RH-0, son muchas las personas que han seguido interesándose con este tema para conseguir avances y obtener experiencia.

Es una motivación importante conseguir que el robot humanoide RH-2 creado en la Universidad Carlos III, sea el más avanzado tecnológicamente, que su caminata sea estable y se comunique con el entorno. Lograr que el robot destaque en ferias internacionales de robótica como lo hace el HRP-3 o incluso el ASIMO y convertirse en el precursor en España, al igual que lo fue el RH-0.

1.4 ESTRUCTURA Y CONTENIDO DE LA MEMORIA

A continuación se hace referencia a los capítulos que componen la memoria del proyecto incluyendo un breve comentario explicando su contenido:

- Capítulo 2. Análisis del estado del arte de los controladores y su evolución. Descripción del proyecto RH y de las características más importantes del futuro prototipo RH-2.
- Capítulo 3. Introducción del controlador de los motores seleccionado presentando sus características más importantes y comparándolo con el utilizado para los humanoides RH-0 y RH1. Descripción del proceso de preparación de la tarjeta (conexionados, cambios de firmware, pruebas, etc). Descripción del lenguaje TML.
- Capitulo 4. Descripción del CANbus y de sus características más importantes. Análisis del protocolo CANOpen y cómo éste se utiliza para implementar programas que controlen los motores conectados a los driver.
- Capítulo 5. Resumen de las conclusiones obtenidas del estudio del driver y de las pruebas realizadas con él. Planteamiento de trabajos futuros y mejoras a desarrollar a partir del trabajo hecho en este proyecto.



Capítulo 2: Estado del Arte

Introducción al campo de los controladores. Consideraciones de los anteriores proyectos RH-0 y RH-1. Estudio cinemático del nuevo proyecto RH-2.

2.1 LA EVOLUCIÓN DE LOS CONTROLADORES

Aunque nuestro controlador está específicamente diseñado para controlar los motores del robot, el proceso evolutivo comienza en el camino de los reguladores en general, desviándose de éste con la aparición de los motores eléctricos y más aún con el desarrollo de la robótica.

2.1.1 Historia del control automático

Los controladores digitales han dearrollado un largo proceso de mejora que se inició, aproximadamente, en el año 300 a.C. cuando los griegos empezaron a tener la necesidad de medir empíricamente el tiempo. Fue entonces cuando apareció el primer dispositivo realimentado, el reloj de agua inventado por el griego Ktesibios en Egipto. Este reloj utilizaba un regulador de flotador que tenía la función de mantener constante el flujo de agua de un primer a un segundo tanque. Según fuese el nivel del segundo tanque así era el tiempo transcurrido.

La Revolución Industrial en Europa logró el primer "boom" del Control Automático. Ésto vino marcado por la invención de molinos de grano avanzados, hornos, calderas, y en mayor medida por la invención del motor de vapor. Estos dispositivos no se podían regular adecuadamente a mano, por lo que surgió una nueva necesidad para los sistemas de control automáticos. Se inventó una nueva variedad de dispositivos de control, como pueden ser los reguladores de flotador, de temperaturas, de presión y dispositivos de control de velocidad como los gobernadores centrífugos que usaban la fuerza centrífuga de las rotaciones del eje del motor para regular la entrada de vapor y variar así la velocidad de giro del motor.

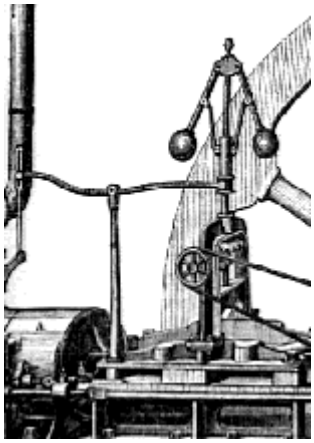


Figura 2.1: Controlador centrífugo de un motor a vapor

Desde finales del sigo XVIII a mediados del XX, se produjo un gran desarrollo en la teoría de la regulación automática, con la evolución paulatina del reloj que permitió desarrollar sistemas más precisos. También influyeron sucesos como la introducción del análisis diferencial, el modelado de sistemas y el avance en cálculos matemáticos.

A principios del siglo XX, hubo una necesidad industrial de instrumentos capaces de medir, grabar y controlar presiones, temperaturas y otras variables, de igual manera que se hace con nuestro controlador. Desde mediados de 1930 Estados unidos ha mantenido un liderazgo sobre otras potencias como Europa en materia de instrumentos controladores y sensores. Los controladores automáticos que se utilizaban eran varios tipos de relé eléctrico y el relé neumático.

Ya en 1922 se había mostrado lo valioso de los controladores PID (Proporcional Integral Derivativo). La función proporcional se conocía desde el comienzo del relé. Sin embargo, la parte integral no se conoció hasta 1920 y la derivativa en 1930.

Otra etapa muy importante en el desarrollo de sistemas de control realimentado fue en la primera mitad del siglo XX con las guerras mundiales. La lucha de los diferentes países por tener las armas e implementos más avanzados, dio paso a un gran auge en la investigación tecnológica. En este periodo el Control Automático no fue la excepción por lo que su desarrollo en sistemas de navegación, aviación y demás fue rápido y conciso.

Durante la Segunda Guerra Mundial, Los Estados Unidos se vieron en la necesidad de desarrollar nuevas ciencias aplicadas incluyendo técnicas e instrumentos avanzados en la industria del control. El campo de la computación se vio afectado por la guerra, en 1946 la escuela de Moore de Ingeniería Eléctrica de la Universidad de Pennsylvania desarrolló el ENIAC (Eletronic Numerical Integrator and Automatic Calculator), el cual fue el primer computador capaz de integrar un sistema simple de ecuaciones diferenciales ordinarias.

Conforme las plantas modernas con muchas entradas y salidas se vuelven más y más complejas, la descripción de un sistema de control moderno requiere una gran cantidad de ecuaciones. La teoría de control clásica que trata de los sistemas con una entrada y una salida, pierde su solidez ante sistemas con entradas y salidas múltiples. Esto se solucionó hacia 1960, gracias a que la disponibilidad de las computadoras digitales hizo posible el análisis de sistemas complejos, la teoría de control moderna, basada en el análisis en el dominio del tiempo y la síntesis a partir de variables de estados. Con esto se consiguió enfrentarse a la creciente complejidad de las plantas modernas y los requerimientos limitativos respecto a la precisión, el peso y el coste en aplicaciones militares, espaciales e industriales.

Ahora que las computadoras digitales se han vuelto más baratas y más compactas, se usan como parte integral de los sistemas de control. Las aplicaciones recientes de la teoría de control moderna incluyen sistemas ajenos a la ingeniería, como los biológicos y los económicos haciendo que el Control Automático sea algo

con lo que se convive día a día, y haciendo que la vida de cada persona sea más fácil.

2.1.2 Los controladores de motores eléctricos

Control de un motor es un término genérico que significa muchas cosas, desde un simple interruptor de paso hasta un complejo sistema con componentes tales como relevadores, controles de tiempo e interruptores. Sin embargo, la función común es la misma en cualquier caso: esto es, controlar alguna operación del motor eléctrico. Por lo tanto, al seleccionar e instalar equipo de control para un motor se debe considerar una gran cantidad de diversos factores a fin de que pueda funcionar correctamente junto a la máquina para la que se diseña.

Para conocer el propósito de los controladores hace falta conocer los factores que hacen que se necesite uno. Algunos de los aspectos a considerar respecto al controlador, al seleccionarlo e instalarlo, pueden enumerarse como sigue:

1) Arranque: El motor se puede arrancar conectándolo directamente a través de la línea. Sin embargo, la máquina impulsada se puede dañar si se arranca con ese esfuerzo giratorio repentino. El arranque debe hacerse lenta y gradualmente, no sólo para proteger la máquina, sino porque la oleada de corriente de la línea durante el arranque puede ser demasiado grande. La frecuencia del arranque de los motores también comprende el empleo del controlador.
2) Paro: Los controladores permiten el funcionamiento hasta la detención de los motores y también imprimen una acción de freno cuando se debe detener la máquina rápidamente. La parada rápida es una función para casos de emergencia.
3) Inversión de la rotación: Se necesitan controladores para cambiar automáticamente la dirección de la rotación. La acción de inversión de los controladores es un proceso continuo en muchas aplicaciones industriales. Esta puede hacerse por medio de estaciones de botones, un interruptor de tambor o un módulo inversor de giro.
4) Marcha: Las velocidades y características de operación deseadas son función y propósito directos de los controladores. Éstos protegen a los motores, operadores, máquinas y materiales, mientras funcionan.
5) Control de velocidad: Algunos controladores pueden mantener velocidades muy precisas para propósitos de procesos industriales, pero se necesitan de otro tipo para cambiar las velocidades de los motores por pasos o gradualmente.
6) Seguridad del operador: Muchas salvaguardas mecánicas han dado origen a métodos eléctricos. Los dispositivos piloto de control eléctrico afectan

directamente a los controladores al proteger a los operadores de la máquina contra condiciones inseguras.

7) Protección contra daños: Una parte de la función de una máquina automática es la de protegerse a sí misma contra daños, así corno a los materiales manufacturados o elaborados. Por ejemplo, se impiden los atascamientos de los transportadores. Las máquinas se pueden hacer funcionar en reversa, detenerse, trabajar a velocidad lenta o lo que sea necesario para realizar la labor de protección.

8) Mantenimiento de los dispositivos de arranque: Una vez instalados y ajustados adecuadamente, los arrancadores para motor mantendrán el tiempo de arranque, voltajes, corriente y torque confiables, en beneficio de la máquina impulsada y el sistema de energía. Los fusibles, cortacircuitos e interruptores de desconexión de tamaño apropiado para el arranque, constituyen buenas prácticas de instalación que se rigen por los códigos eléctricos.

2.1.3 Evolución en el control de los motores

Tradicionalmente, el control de motores estaba ceñido de una mayor manera, a términos como el arranque, frenado, reversión del movimiento y aceleración del motor. Dicho control se realizaba mediante elementos meramente eléctricos como pueden ser relés, interruptores, etc. Sin embargo, con el desarrollo de los computadores y de los componentes electrónicos en general, en la actualidad el concepto de control de motores eléctricos, no sólo se refiere a los dispositivos eléctricos convencionales, también a dispositivos electrónicos, cuyo estudio se relaciona con la llamada electrónica de potencia, lo cual da un mayor grado de complejidad a los circuitos de control y en consecuencia las funciones que estos pueden conseguir se diversifican adaptándose a todos los medios en los que se utilizan los motores eléctricos.

Actualmente, el control de los motores se puede ejecutar desde ordenadores o PLCs. Esto es realmente importante en robótica, en medios donde se utilicen robots industriales, etc., en los que se necesite controlar cada uno de los motores de forma independiente y que además cada grado de libertad ejecute un perfil distinto de forma simultánea.

Un avance más en este proceso se consigue cuando la generación de trayectorias ya no se calcula en la unidad central de control, sino que son los llamados drivers o controladores los que se encargan de esta tarea. Suelen ser tarjetas anexas al motor que además de tener incorporados pequeños microprocesadores y memorias en las que tener almacenados parámetros de configuración que utilice el driver, pueden llegar a almacenar programas y funciones que sean ejecutados mediante llamadas desde la CPU.

2.1.4 La evolución de los controladores en la robótica

En el campo de la robótica, se pueden destacar dos métodos de control del movimiento de las articulaciones de un robot. El primero de ellos es mediante el uso de un circuito de control o driver con el cuál se proporcionan la tensión e intensidad de alimentación de forma que se puede controlar el sentido y la velocidad de giro. Un driver es un amplificador eléctrico especial usado normalmente para controlar motores de corriente continua. El driver controla las señales de realimentación y continuamente se adapta para corregir la desviación del comportamiento esperado.

El driver recibe una señal de control de una CPU, la amplifica y transmite la corriente eléctrica a un motor eléctrico para producir el movimiento proporcional a la señal de control. Típicamente la señal de control representa una velocidad deseada, pero también puede significar un momento de rotación o una posición. Un sensor de velocidad o de posición conectado al motor envía la velocidad o posición real del motor al driver. Entonces el driver compara la velocidad o posición real del motor con la velocidad o posición pedida de motor y cambia la frecuencia de voltaje al motor para corregir para cualquier error en la velocidad o la posición.

Los driver se encargan de la generación de trayectorias, y los más modernos pueden almacenar y ejecutar programas y otra multitud de opciones, lo que les convierte en pequeños microprocesadores subordinados a la unidad de control. Así se consigue una descentralización en el robot. Otra forma de aumentar el grado de descentralización del sistema es introducir varios procesadores que trabajen conjuntamente pudiéndose así ejecutar varios procesos simultáneamente.

El segundo de ellos se consigue mediante el uso de servomotores. Un servomotor (también llamado Servo) es un dispositivo similar a un motor de corriente continua, que tiene la capacidad de ubicarse en cualquier posición dentro de su rango de operación (comúnmente entre 0° y 180°) y mantenerse estable en dicha posición. Está conformado por un motor, una caja reductora, una retroalimentación y un circuito de control. Los servos se utilizan frecuentemente en sistemas de radiocontrol y robótica, pero su uso no está limitado a estos. Es posible modificar un servomotor para obtener un motor de corriente continua que, si bien ya no tiene la capacidad de control del servo, conserva la fuerza, velocidad y baja inercia que caracteriza a estos dispositivos.

El control de un servo se reduce a indicar su posición mediante una señal cuadrada de voltaje, donde el ángulo de ubicación de la flecha depende de la duración del nivel alto de la señal. Cada servo motor, dependiendo de la marca y modelo utilizado, tiene sus propios márgenes de operación.

Normalmente son controlados por un microprocesador que ejecuta todas las tareas. Este microprocesador se encarga de generar las trayectorias y movimientos de cada motor y se tiene una comunicación directa con cada uno. Por lo tanto se requiere de un cableado y comunicación con los servomotores mucho más complejo.

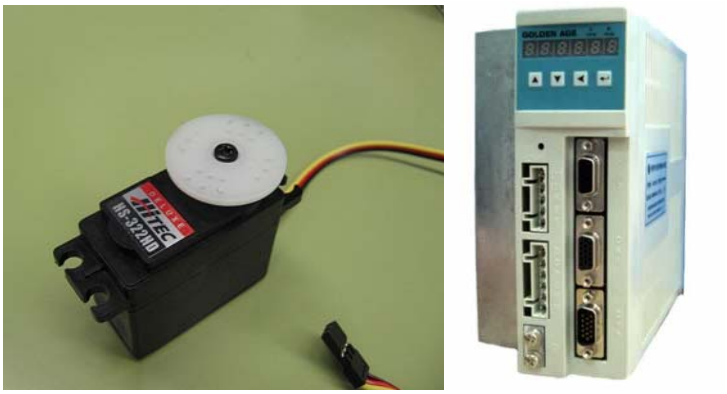


Figura 2.2: Servomotor y driver

Como ya se dijo antes, el uso de drivers hace que se genere un modelo distribuido en el que la unidad de control les deriva funcionalidades, y el modelo de los servomotores genera un modelo centralizado que se encarga del control y generado de trayectorias de todos los servos.

Las ventajas de los sistemas distribuidos con respecto de los centralizados son:

- Datos compartidos: Un sistema distribuido permite que varios usuarios (en nuestro caso cada microprocesador) tengan acceso a una base de datos común.
- Dispositivos compartidos: De igual manera, se pueden compartir un mismo periférico entre distintos microprocesadores.
- Comunicación: un sistema distribuido facilita la comunicación entre computadoras aisladas, por ejemplo mediante conexión Lan-Internet, inalámbrica.
- Flexibilidad: Un sistema distribuido difunde la carga de trabajo entre los distintos microprocesadores del sistema.

Las desventajas de los sistemas distribuidos son:

- Software: No hay mucha experiencia en el diseño, implantación y uso del software distribuido. Existe poco software para los sistemas distribuidos en la actualidad.
- Redes: si el sistema llega a ser dependiente de la red, la pérdida o saturación de ésta puede bloquear el sistema.
- Seguridad: si las personas pueden tener acceso a los datos en todo el sistema, entonces también pueden tener acceso a parámetros de configuración críticos y datos importantes quedan vulnerables

El desarrollo en cada tipo de sistema de control viene dado por distintos fenómenos. En el caso del sistema centralizado, es la evolución de la unidad de control la que marca el grado evolutivo. Factores como la velocidad del procesador son indicadores de ello.

Sin embargo, en el caso del sistema distribuido aunque los microprocesadores influyen en el rendimiento del robot, es el circuito que controla el motor el que marca el grado de desarrollo. Cuantos más procesos se puedan delegar al driver, más liberada quedará la unidad de control y menos se saturará el bus de datos.

Esta tabla refleja la evolución de los procesadores de algunos de los robots más significativos:

FECHA	ROBOT	CPU
1997	P3	2x Microspec II 1100 MHz
1997	WABIAN-RIV	Intel MMX Pentium 200MHz
2000	ASIMO	Desconocido
2003	QRIO	3x64 bit RISC
2003	HOAP-3	Wireless: Geode GX1 Main: Pentium II 700 MHz
2003	H7	2xPentium IV 3 GHz
2006	WABIAN 2R	Pentium M 1.6GHz
2007	HRP-3	2x32bit RISC-CPU SH4 240[MHz]

Tabla 2.1: Procesadores de robots significativos

La siguiente figura nos muestra imágenes de los robots analizados:

Figura 2.3: Imágenes de los robots analizados

2.1.5 Los drivers del mercado

En este apartado vamos a analizar los distintos driver que se pueden encontrar en el mercado y vamos a mostrar los catálogos que tienen algunas empresas para comprender como son los controladores actualmente y buscar entre ellos el que mejor se adapte a nuestras necesidades.

En el mercado hay multitud de compañías que ofrecen drivers para controlar los motores. El catálogo que ofrecen suele ser bastante amplio para cubrir todas las necesidades que pueda tener el cliente. Los hay para controlar motores trifásicos, de corriente continua, paso a paso, sin escobillas, etc. Las potencias que pueden suministrar van desde decenas de vatios a miles, ofrecen distintos protocolos para comunicaciones como pueden ser CANOpen, RS-232 o Ethernet entre muchos y el tamaño también puede variar significativamente entre modelos y compañías.

- Elmo Motion

Los drivers digitales de Elmo usan la tecnología de control de movimiento SimplIQ desarrollada por Elmo, controlan motores de corriente continua y alterna brushless y destacan una gama de funcionalidades incluyendo la posibilidad de conexión a red CANOpen para un control distribuido. Las series Tweeter, Whistle, Guitar, Harmonica, Bassoon, Cello, Drum, Cornet y Tuba comprenden los productos principales en la gama de Elmo de servo drives digitales, combinando la alta densidad de potencia, la funcionalidad inteligente y el diseño espacial amistoso.

Los drivers de Elmo Motion se adecuan para ambientes donde se requiere control de movimientos de forma distribuida o comunicaciones en tiempo real. Estos incluyen los protocolos CANOPEN DS 301 y una puesta en práctica de los protocolos DS 402 Y DS 305. Los drivers incorporan un traductor binario y ello permite una comunicación rápida entre el maestro y el controlador de forma simultánea a través de los puertos RS-232 y CAN. Tienen varios modos de control como son el de posición, velocidad, corriente, etc. y permiten lazo doble de realimentación

Feature	Units	Tweeter	Whistle	Harmonica	Cello	Guitar	Drum	Bassoon	Cornet	Tuba
Supply Voltage range	VDC	7.5 ~ 95	7.5 ~ 95	10 ~ 195	10 ~ 195	11~195	11 ~ 390			
Supply Voltage range	VAC							1X30 ~ 270	1x60 up to 3 x 505	1x60 up to 3 x 505
Motor		DC Brush, Sinusoidal, trapezoidal								
Operating Modes		Current, Velocity, Position & Advance Position								
Commands		Analog, PWM, Pulse and Direction, Software Commands								
Feedbacks		Incremental Encoder, Resolver, Digital Halls, Analog SIN-COS, Absolute, Analog Halls, Tacho, Potentiometer								
Cont. Output Current	A	2.5 ~ 3.3	1 ~ 20	2.0 ~ 13.3	2.25 ~ 30	3~45	18 ~ 90	1 ~ 6	1.4 ~ 9	12 ~ 20
Output Power Range	KW	0.16 ~ 0.2	0.05 ~ 1.60	0.20 ~ 1.10	0.24 ~ 3.40	0.48~4.8	2.7 ~ 9.6	0.32 ~ 1.90	0.42 ~ 3.40	3.60 ~ 11.30
Digital In, Out, Analog In		6/2/1	6/2/1	6/2/1	10/5/2	6/2/1	6/2/1	6/2/1	10/6/2	10/6/2
Communications		RS-232, CANOpen DS 301, DSP 305 and DSP 402								

Figura 2.4: Características de los drivers de ELMO

- Technosoft Motion

Los servo drivers de Technosoft pueden dirigir motores brushless de corriente alterna, brushless de corriente continua, motores brushed de corriente continua y motores paso a paso pudiendo realizar el control mediante modos de control como la posición, la velocidad o el torque. Estos driver usan órdenes de movimiento de alto nivel y ofrecen funcionalidad PLC. Los canales de comunicación que tienen son el RS-232 y CANOpen.

Estos drivers inteligentes tienen un lenguaje de programación llamado TML (Technosoft Motion Language) que contiene instrucciones que permiten definir y comenzar secuencias de movimiento complejas desde el maestro, o ejecutar secuencias de movimiento prealmacenadas seleccionadas a través de las líneas de entrada /salida, de forma independiente.

Intelligent Drive / Motor	Supply Voltage [V]	Output Current [A] Continuous	Peak	Output Power [W]	Usable with Motor Types
Drives for brushless / step / DC / linear motors					
IDM680 (CAN / CANopen)	12 - 80	8	16.5	640	Brushless sinusoidal, trapezoidal, DC brush, stepper
IDM240 / IDM640 (CAN / CANopen)	12 - 48 12 - 80	5 8	16	240 640	Brushless sinusoidal, trapezoidal, DC brush, stepper
IDM3000 (CAN / CANopen)	160 - 325	10	30	3000	Brushless sinusoidal, trapezoidal, DC brush, induction
TMC-3D	12 - 80	8	16.5	640	Brushless sinusoidal, trapezoidal, DC brush, stepper
ISM4803 (CAN / CANopen)	12 - 48	3	6	144	Brushless sinusoidal, trapezoidal DC brush, stepper
ISCM4805 / ISCM8005 (CAN / CANopen)	12 - 48 / 12 - 80	5	16	240 400	Brushless sinusoidal, trapezoidal, DC brush, stepper
IBL3605 NEW (CAN / CANopen)	12 - 36	5	16	180	Brushless sinusoidal, trapezoidal, DC brush, stepper
IBL2403 (CAN / CANopen)	12 - 28	3	6	84	Brushless sinusoidal, trapezoidal, DC brush, stepper
IBL2401 (CAN / CANopen)	6 - 24	1	3	25	Brushless sinusoidal or trapezoidal, DC brush, stepper
PIM2403 (CAN / CANopen)	12 - 24	3	6	84	Brushless sinusoidal, trapezoidal, DC brush, stepper
PIM2401 (CAN / CANopen)	6 - 24	1	3	25	Brushless sinusoidal or trapezoidal, DC brush, stepper
Minidrives for step / DC motors					
IPS110 (CAN / CANopen)	12 - 45	0.5	1	25	Stepper (also, DC brush)
IPS210 (CAN / CANopen)	8 - 24	0.5	1	2 x 12	Stepper (also, DC brush)
Intelligent motors					
IM23x (CAN / CANopen)	12 - 48	1 - 3	3 - 9	40 - 120	Embedded intelligent brushless motor family
IS23x (CAN / CANopen)	12 - 48				Embedded step motor family

Figura 2.5: Características de los driver de Technosoft

- Fiveco

Fiveco ofrece una pequeña gama de drivers que se conectan mediante Ethernet, solo pueden manejar motores con/sin escobillas DC y realizan el control solamente mediante la posición o la velocidad.

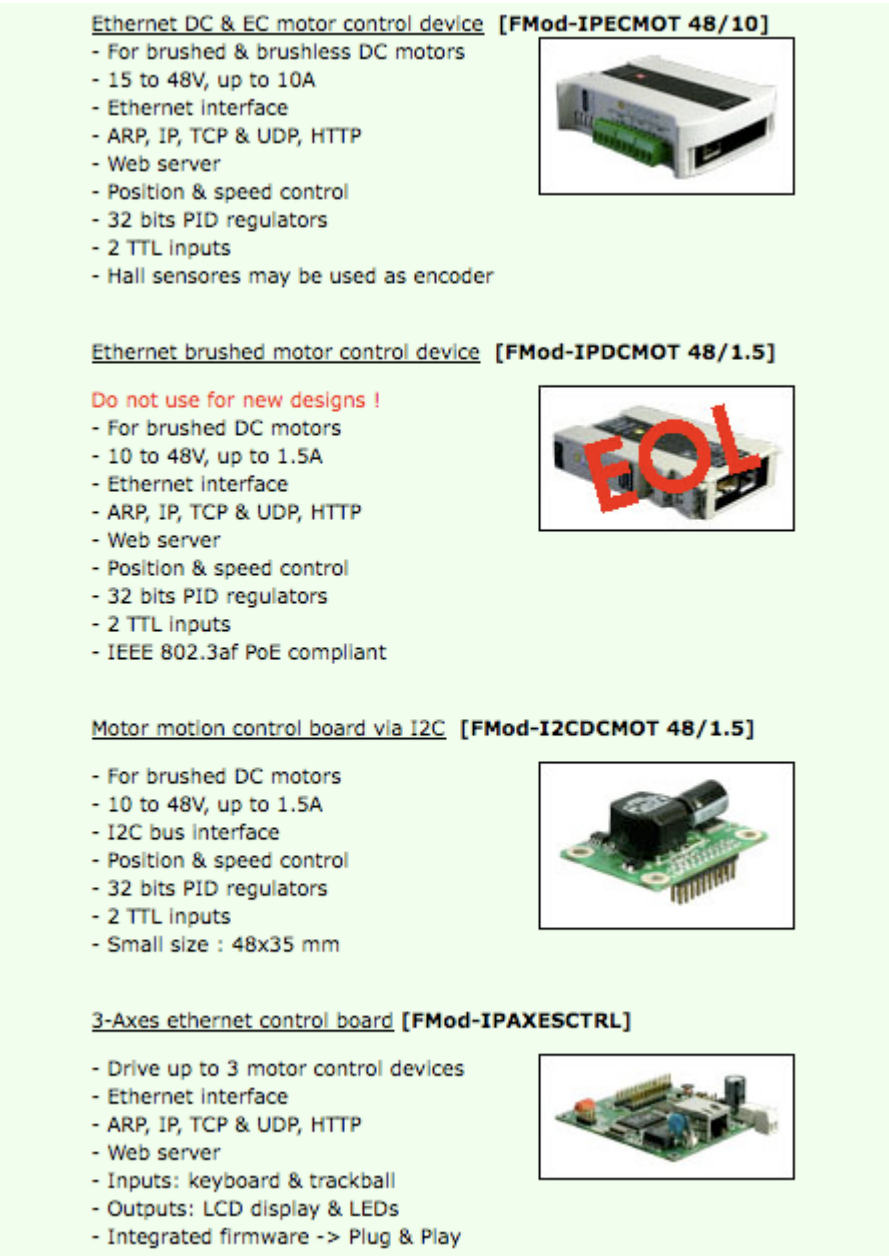


Figura 2.6: Características de los driver de Fiveco

- Maxon Motor

Los controladores Maxon son capaces de controlar motores de corriente continua y alterna. Son servoamplificadores de 1 a 4 cuadrantes. Pueden realizar el control de los motores mediante corriente, velocidad (bien mediante un tacho o gracias a un encoder) o el voltaje. Pueden usar CANOpen para las comunicaciones y tienen desarrollado un EPOS (Easy-to-use Positioning) con interface CANOpen como unidad de control de posición.

Electrical Data	LSC 30/2
Supply voltage V_{cc}	12 - 30 VDC
Max. output voltage	V_{cc} - 5 V
Max. output current I_{max}	2 A
Continuous output current I_{cont}	2 A
Mechanical data	
Weight	approx. 330 g
Dimensions (LxWxH)	103x100x34 mm
Mounting	Flange for M4-screws

Electrical Data	ADS 50/5	ADS 50/10
Supply voltage V_{cc}	12 - 50 VDC	12 - 50 VDC
Max. output voltage	0.9 x V_{cc}	0.9 x V_{cc}
Max. output current I_{max}	10 A	20 A
Continuous output current I_{cont}	5 A	10 A
Mechanical data		
Weight (approx.)	400 g	400 g
Dimensions (LxWxH)	180x103x26 mm	180x103x26 mm
Mounting	Flange for M4-screws	

Figura 2.7: Características de algunos driver de Maxon

Al ser fabricantes de motores, también ofrecen una amplia gama de accesorios tales como sensores para la realimentación reductores, cables, etc. Además los drivers están especialmente diseñados para el control de motores Maxon.

Electrical Data	ADS_E 50/5	ADS_E 50/10
Supply voltage V_{cc}	12 - 50 VDC	12 - 50 VDC
Max. output voltage	0.9 x V_{cc}	0.9 x V_{cc}
Max. output current I_{max}	10 A	20 A
Continuous output current I_{cont}	5 A	10 A
Mechanical data		
Weight (approx.)	175 g	410 g
Dimensions (LxWxH)	160x100x16 mm	160x100x30.5 mm
Mounting		Rack-Installation

Electrical Data	AECS 35/3
Supply voltage V_{cc}	8 - 35 VDC
Max. output voltage	≙ V_{cc}
Max. output current I_{max}	5 A
Continuous output current I_{cont}	3 A
Mechanical data	
Weight	approx. 20 g
Dimensions (LxWxH)	74x51x20 mm
Mounting	4 hexagonal M3 distance pins with inner winding

Figura 2.8: Características de otros driver de Maxon

2.1.6 Conclusiones

Este es un pequeño análisis de algunas de las empresas dedicadas a la fabricación de drivers. Como resulta obvio, hay muchísimas más empresas en el sector pero aquí sólo se ha querido resaltar las que ofrecen una serie de productos que se adecuan de una mayor manera a nuestras necesidades.

De este catálogo se procederá a analizar dos driver en particular. Uno es el Harmonica de Elmo Motion, pues se utilizó en los prototipos RH-0 y RH-1. El otro es el ISCM8005 de Technosoft ya que después de analizar los catálogos de mercado y viendo sus características, hemos llegado a la conclusión de que puede ser un buen candidato como controlador de todos los grados de libertad del RH-2. Ambos drivers serán analizados más en profundidad a lo largo del proyecto, haciéndose una comparación de ambos en el apartado 3.2.2 de este proyecto para ver si la tarjeta ISCM8005 es en verdad un buen sustituto del driver de Elmo Motion.

2.2. EL PROYECTO RH

El proyecto de robots humanoides RH de la Universidad Carlos III consta de dos prototipos ya construidos: RH-0 y el RH-1. El RH-1 suponía simplemente la modificación o sustitución de algunos elementos hardware y rediseños mecánicos del RH-0. Excepto algunos cambios estructurales, se mantiene el tamaño y los grados de libertad. Ambos han sido desarrollados tomando como modelos los prototipos más avanzados existentes en aquel momento, como eran el ASIMO de Honda y el HRP-2P de Kawada (que ya ha quedado obsoleto al salir el modelo HRP-3P). Se han definido como un sistema mecánico de 21 grados de libertad (GDL o DOF), todos ellos activos (23 contando la cámara). En cada GDL hay un motor encargado de su movimiento, y están dispuestos de modo que le dotan de apariencia humana tanto en el aspecto como en la capacidad de locomoción, con las limitaciones propias de la extrema complejidad del sistema locomotor de un ser humano. El proyecto se dividió en varios grupos de trabajo, y en este apartado se mostrará el estado de los trabajos de diseño de arquitectura, cinemático, mecánico, dinámico y de software.

2.2.1. Diseño de la estructura cinemática

El resultado final de diferentes estudios y configuraciones, es la distribución de todos sus grados de libertad que se muestra en la tabla 2.2 y en la figura 2.9:

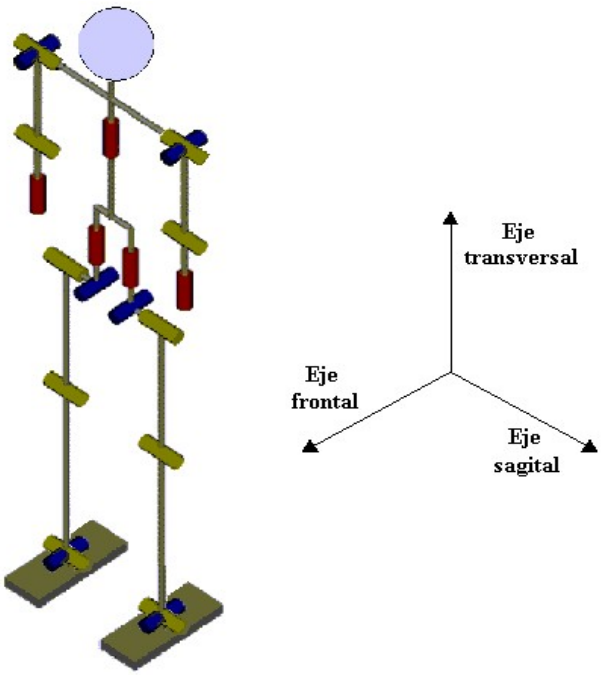


Figura 2.9: Estructura cinemática de los Robots Humanoides RH-0 y RH-1

GDL	Número	Eje de movimiento
PIERNAS	12 GDL's	
Cadera	3 (x2)	Sagital Frontal Transversal
Rodilla	1 (x2)	Sagital
Tobillo	2 (x2)	Sagital Frontal
BRAZOS	8 GDL's	
Hombros	2 (x2)	Sagital Frontal
Codo	1 (x2)	Sagital
Muñeca	1 (x2)	Transversal
TRONCO	1 GDL	
Tronco	1	Transversal
CABEZA	2 GDL's	
Cámara	2	Transversal (izquierda – derecha) Sagital (arriba – abajo)
TOTAL	23 GDL's con cámara (21 GDL's sin cámara)	

Tabla 2.2: Distribución de GDL's de los Robots Humanoides RH-0 y RH-1

En base a este diseño, el robot será capaz de:

1) Asistir a las personas en sus entornos cotidianos
2) Caminar en línea recta y voltear a los lados
3) Subir y bajar escaleras
4) Simular al caminar el movimiento de los brazos de una persona
5) Coger objetos de peso inferior a 0.5 Kg.
6) Gesticular con los brazos: señalar, saludar, etc.

2.2.2. Diseño Mecánico

Los Robots Humanoides RH-0 y RH-1 son unos sistemas mecánicos de 23 Grados de Libertad (21 si no tenemos en cuenta los de la cámara). Se distribuyen de la siguiente forma por sus extremidades:

- Piernas:

Dispone cada una de 6 GDL's distribuidos entre el tobillo, la rodilla y la cadera. La cadera posee 3 GDL's, uno en el plano sagital, otro en el frontal y el tercero en el plano transversal, utilizado en el cambio de dirección del movimiento; la rodilla tiene 1 GDL en el plano sagital, pues no es necesario ningún otro; y el tobillo posee 2 GDL's, en los planos sagital, para adaptar el pie al suelo, y frontal que permite el balanceo junto con el de la cadera para mantener el equilibrio.

- Brazos:

Cada brazo tiene 4 GDL's distribuidos entre el hombro, el codo y la muñeca. En el hombro existen 2 GDL's en los planos sagital y frontal, en el codo hay 1 GDL's en plano sagital y en la muñeca existe únicamente 1 GDL en el plano transversal. Esta distribución permite la manipulación de objetos, aunque no le confiere la movilidad de un brazo humano.

- Tronco y Cabeza:

El tronco posee 1 GDL en plano transversal, que le permite el giro en ese plano sin tener que mover las piernas. La cabeza posee 2 GDL's; uno para girar hacia los lados y otro que permite el giro hacia arriba o hacia abajo.

Las siguientes figuras muestran diferentes características a tener en cuenta para las siguientes fases del Proyecto, como son el desarrollo del Modelo Dinámico y la configuración del Sistema Hardware de Control. Los datos que aquí se aportan son la descripción dimensional, mostrando también el diseño mecánico completo y el prototipo ya construido.

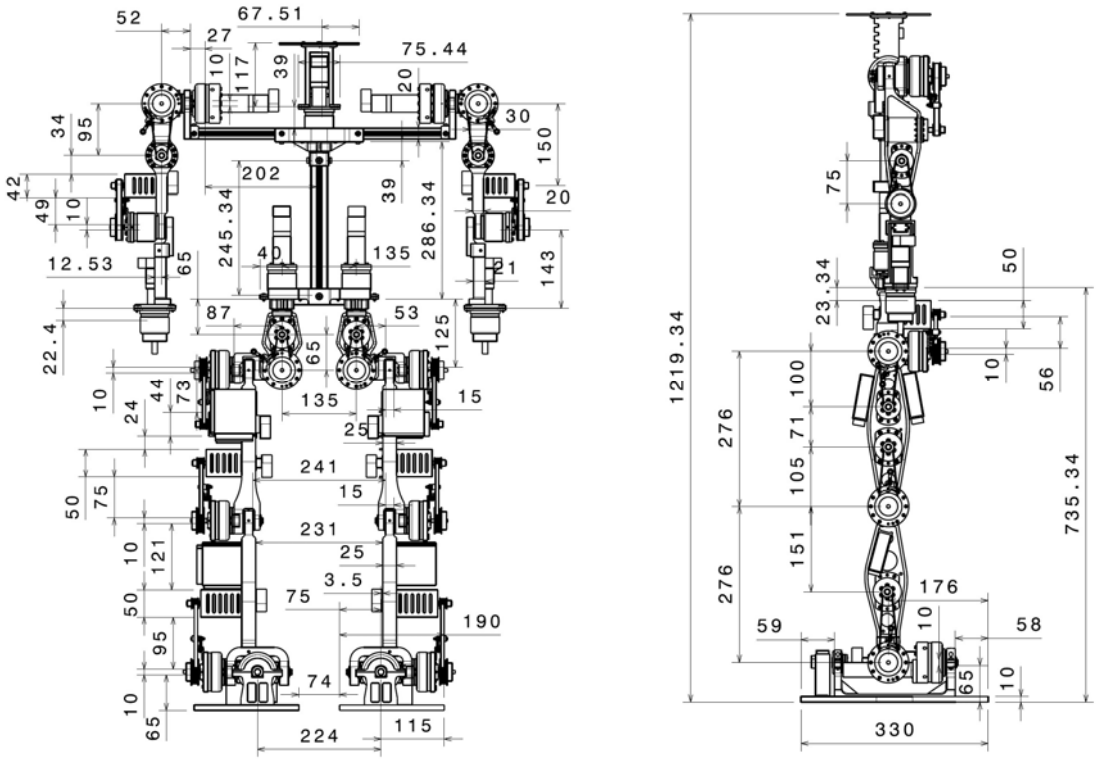


Figura 2.10 Imágenes del diseño mecánico completo

El resultado final del diseño es el prototipo construido:

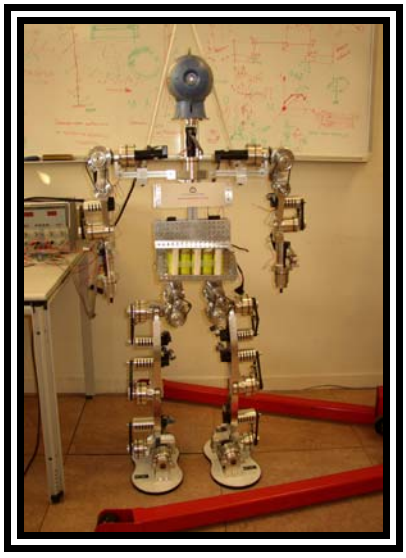


Figura 2.11 Prototipo del Robot Humanoide RH-0

2.2.3 Caminata

El diseño cinemático se realizó para la tarea CAMINAR EN LÍNEA RECTA, de forma estable con un movimiento adecuado de las piernas, por lo que sólo se muestra el estudio cinemático del tren inferior. Se han realizado dos modelos que recibirán tratamientos distintos, con el objetivo de dividir el problema en otros más simples para después integrarlos. Estos modelos son:

- Apoyo doble: la cadena cinemática de los eslabones de la pierna está cerrada
- Apoyo simple: la cadena cinemática de los eslabones de la pierna está abierta

Según se muestra en la figura 2.10 el movimiento se ha dividido en 3 fases, que a su vez se dividen en dos subfases: apoyo doble, con balance del centro de gravedad (son las fases 1.1, 2.1, 3.1 y 3.3), y apoyo simple, avance del robot (son las fases 1.2, 2.2 y 3.2). Se puede observar en todas las fases del movimiento, mediante la trayectoria de su proyección sobre el suelo, que el ZMP (Zero Moment Point) se mantiene dentro de la zona estable. La figura 2.15 muestra también la longitud del paso (Lp).

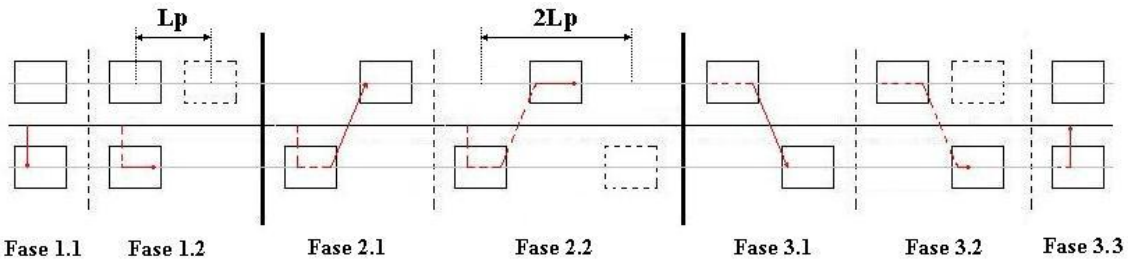


Figura 2.12: Fases del movimiento

2.2.4 Diseño del Sistema Software

El desarrollo del sistema software viene determinado en su primera fase por la definición de los requerimientos básicos. Los que se han propuesto son los siguientes:

- Software distribuido: se deben cumplir requerimientos de tiempo real. Se deben poder realizar cálculos pesados sin necesidad de ejecución en tiempo real en la estación de trabajo.

- Requerimiento de reactividad: el robot debe ser capaz de reaccionar y actuar en consecuencia de modo que pueda integrarse en entornos no estructurados y dinámicos.
- Capacidad de aprendizaje: funcionalidad muy importante a añadir para su integración en entornos dinámicos y no estructurados.

Una vez definidos estos requerimientos, se ha diseñado una estructura básica dividida en 3 niveles o capas, tal y como muestra la siguiente figura:

Figura 2.13: Distribución del Software en capas

La que se muestra es una división jerárquica, de mayor a menor inteligencia y de menor a mayor detalle. Las capas están interconectadas entre sí a través de comunicaciones inalámbricas (capas de organización y coordinación) y del bus de campo CANbus (capas de coordinación y ejecución). Estos niveles se detallan a continuación, así como la interacción entre sus elementos.

El siguiente cuadro refleja las variaciones que hubo entre los dos prototipos:

ACCIONADORES	RH-0	RH-1
motor	Faulhaber 24 V DC	Faulhaber 24 V DC
frenos motor	MBZ 22 de Faulhaber	MBZ 22 de Faulhaber
SENSORIZACION		
encoder	HEDS 5540 A de Faulhaber.	HEDS 5540 A de Faulhaber.
giróscopo	Silicon CRS07	Silicon CRS07
inclinómetro	ADXL Analog Devices	ADXL Analog Devices
sincronismo	contrinex serie 620	contrinex serie 620
COMPUTACIÓN		
microprocesador	SECO M570 PC/104 400MHz	Pentium M PC/104 Digital Logic
driver	elmo HARMONICA A5/50 CAN	elmo HARMONICA A5/50 CAN

Tabla 2.3: Comparativa de los elementos de los prototipos RH-0 y RH-1

Se han realizado dos cambios para la mejora del hardware del sistema: cambio del PC y cambio de la placa del controlador. Las ventajas del nuevo PC con respecto al antiguo son: mayor capacidad, bus más avanzado (PC\104+) y por lo tanto mayor velocidad. El bus datos del PC actual es más parecido a los PC de uso normal. El nuevo controlador tiene la ventaja de que puede trabajar en modo de interrupciones.

2.2.5 HARMONICA A5/50 CAN

Al ser el objetivo de este proyecto la mejora de la parte del sistema de control de movimientos, se procede a analizar más en profundidad el driver usado en los prototipos RH-0 y RH-1. Así se tendrá un punto de partida a la hora de buscar un sustitutivo del driver que sea mejor en sus características tanto técnicas como a nivel de programación (funciones, modos de control, etc) y por ello se adapte de una forma más adecuada a nuestras necesidades.

El driver utilizado en los prototipos anteriores fue el HARMONICA A5/50 CAN de ELMO MOTION CONTROL. Las características generales del dispositivo, agrupadas esquemáticamente, son las siguientes:

- Modos de Control de Movimiento:
  - control de Corriente/Par (velocidad en bucle cerrado hasta 14 KHz).
  - control de Velocidad (velocidad en bucle cerrado hasta 7 KHz).

  - control de Posición (velocidad en bucle cerrado hasta 3.5 KHz).
- Modos avanzados de control de movimientos:
  - PTP, PT, PVT, ECAM, Lazo Dual (con encoder auxiliar).
  - Captura rápida de Entradas.
- Filtración Avanzada. Programación de Ganancias:
  - Programación de ganancias en marcha en modo de control de corriente y de velocidad.
  - Control de velocidad y de posición con reguladores PI.
  - Conmutación automática de los motores trifásicos.
- Totalmente programable:
  - Estructura de programa con órdenes de movimiento.
  - Posibilidad de configurar una parada automática ante fallos de posición que superen un error prefijado vía software.
  - Posibilidad de configurar señales externas que permiten empezar, parar o modificar un movimiento.
  - Programación basada en eventos.
- Opciones de realimentación:
  - Encoder Incremental (hasta 20 Millones de cuentas).
  - Sensores Hall - hasta 2 KHz.
  - Encoder Incremental con Sensores Hall para conmutación (hasta 20 Millones de cuentas).
  - Encoder absoluto.
  - Resolver.
  - DC tacómetro para el bucle de control de la velocidad.
  - Potenciómetro analógico para el bucle de control de posición.
  - Tensión de alimentación para todas las opciones de feedback.
- Entradas / Salidas
  - Indicación de fallos.
  - Entrada analógica con resolución de 12 bits.
  - Interruptores de límites.
  - Paradas de emergencia.
  - Entradas / Salidas digitales programables con aislamiento óptico (6 entradas, 2 salidas).
  - Entradas capturadas de eventos rápidas (2).
- Opciones de comunicación.

- RS232.
- CANOpen

Figura 2.14: HARMONICA A5/50 CAN de Elmo

2.3 EL RH-2

El nuevo robot dispone de 24 grados de libertad y se estima un peso de 60 Kg y una velocidad de 1 Km/h durante la caminata. Se calcula que podrá transportar objetos de 2 Kg de peso e incluso sentarse. Su altura aumenta de 1.2 m a 1.65 m, dotando al robot de un tamaño más acorde al de un humano.

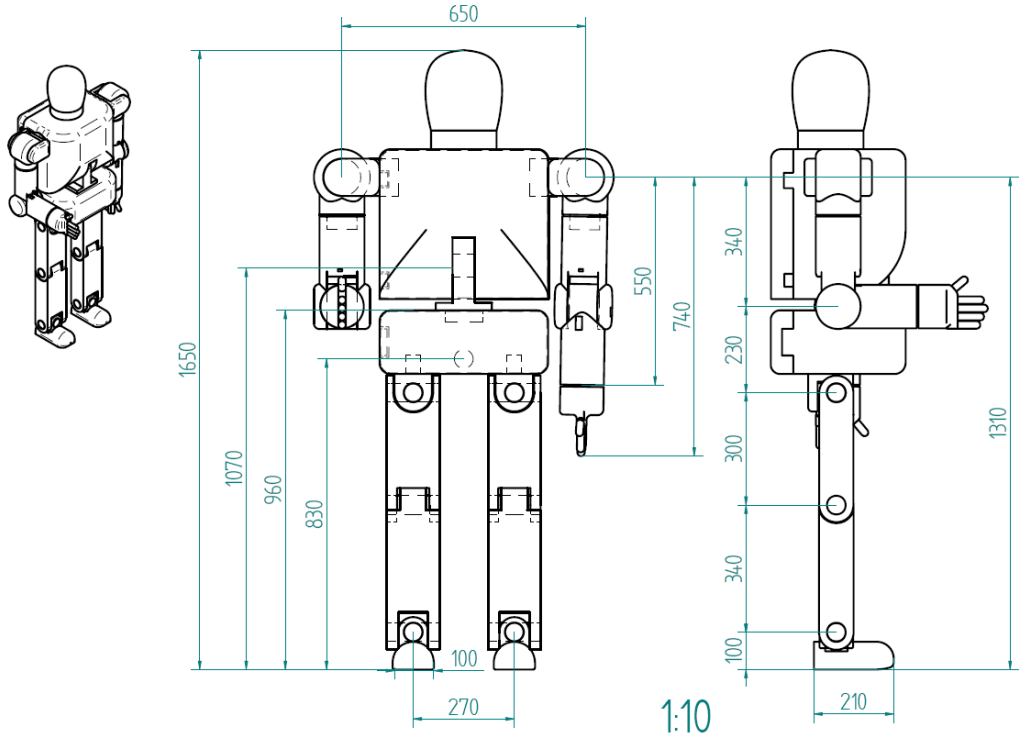


Figura 2.15: Medidas del prototipo RH-2

Los anteriores modelos disponían de 21 grados de libertad. El RH-2 pretende incluir tres grados más de libertad. Uno en cada codo en el plano transversal, que permite al brazo realizar movimientos más parecidos a los humanos. Otro grado de libertad es necesario en el tronco, en su plano frontal, para poder controlar de manera más rápida el balanceo hacia delante y atrás del cuerpo y lograr mantener su centro de masa centrado. Este nuevo grado de libertad dota al robot de la capacidad de plegar más el tronco y poder sentarse.

El Robot Humanoide RH-2 es un sistema mecánico de 24 Grados de Libertad (26 si tenemos en cuenta los motores de la cabeza). Se distribuyen de la siguiente forma por sus extremidades:

- Piernas:

Dispone cada una de 6 GDL's distribuidos entre el tobillo, la rodilla y la cadera. La cadera posee 3 GDL's, uno en el plano sagital, otro en el frontal y el tercero en el plano transversal, utilizado en el cambio de dirección del movimiento; la rodilla tiene 1 GDL en el plano sagital, pues no es necesario ningún otro; y el tobillo posee 2 GDL's, en los planos sagital, para adaptar el pie al suelo, y frontal que permite el balanceo junto con el de la cadera para mantener el equilibrio.

- Brazos:

Dispone cada uno de 5 GDL's distribuidos entre el hombro, el codo y la muñeca. En el hombro existen 2 GDL's en los planos sagital y frontal, en el codo hay 2 GDL's en plano frontal y transversal. En la muñeca existe únicamente 1 GDL en el plano transversal. Esta distribución permite la manipulación de objetos, e intenta que la movilidad sea la de un brazo humano.

- Tronco:

El tronco posee 2 GDL en plano transversal, que le permite el giro en ese plano sin tener que mover las piernas y otro en plano frontal, que le permite regular su inclinación.

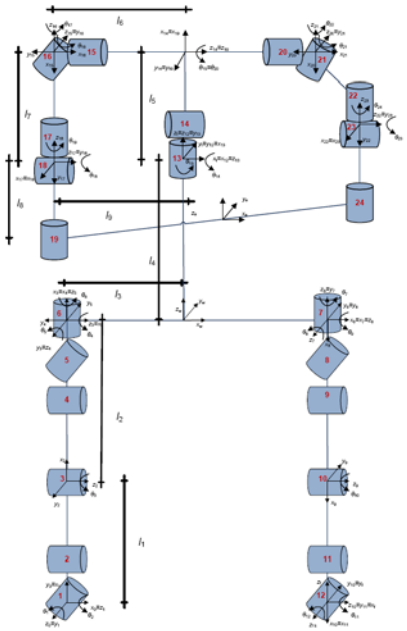


Figura 2.16: Esquema completo de los GDL del RH-2

En un futuro se estudiarán los grados de libertad de la cabeza. Una opción es moverla gracias a nuevas cámaras autodirigidas que incluyen motores para enfocar al elemento en movimiento.

Es necesario un sistema para medir el error de posición entre la posición leída del motor, y la posición real a la salida de las transmisiones. En los proyectos anteriores sólo se recurría a leer el encoder del motor, y no se tenían datos de la posición en la que se encontraba realmente cada articulación. La implementación de los sensores de sincronismo para realizar el homing y buscar la posición de referencia de los motores, no se llevó a cabo. Será necesario introducir otro encoder en cada transmisión y diseñar el circuito de control y adquisición de los datos. Otra de las medidas necesarias es la fuerza que ejercen los pies sobre el suelo. En los proyectos anteriores no se incluían estos sensores. Sin este parámetro y sin el error de posición de las articulaciones no se puede completar el ciclo de control de la caminata.

En la medida de lo posible, el ciclo de control de la caminata del robot debe cerrarse en el mínimo tiempo posible. Es imprescindible que se pueda realizar este lazo de control; el robot debe funcionar en un lazo cerrado para mantener siempre la estabilidad. Recurrir a sistemas lo más distribuidos posibles nos aseguran tiempos de cómputo más reducidos. Se debe demostrar la viabilidad del protocolo CANbus utilizado en los anteriores robots, y comprobar que el hardware que gestiona esta red es el más apropiado.

El montaje del robot en su totalidad es una tarea compleja y es difícil que todo funcione perfectamente. Por eso se quiere realizar una primera fase del proyecto RH-2 para construir la parte inferior, las piernas, y construir un andador.

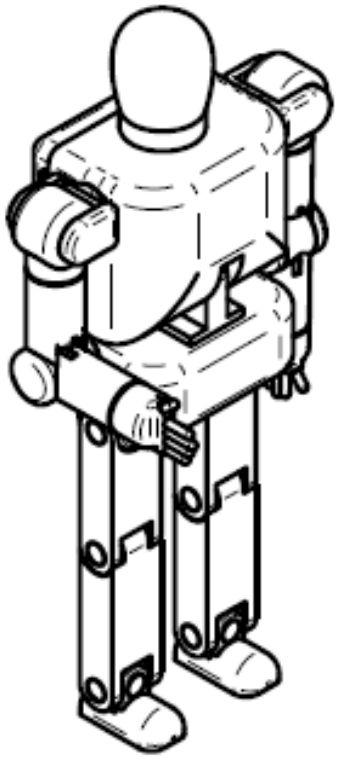
Una vez probado el andador y conseguida una caminata muy estable se dará paso a la segunda fase. Se diseñará concretamente y llevará a cabo la parte superior, haciendo hincapié en la relación del robot con el entorno. Para ello se definirán los sensores exoceptivos necesarios, cámaras, micrófonos, etc., y se estudiará la opción de incluir un microprocesador específico para el tratamiento de imagen y sonido.

Los microprocesadores usados anteriormente se basaban en el protocolo ISA para la comunicación interna. Estos dispositivos fueron problemáticos y ahora se estudiarán distintos protocolos internos como PCI y se buscarán otras opciones de microprocesadores.

El robot no dispone de demasiado espacio para albergar todo el Hardware que requiere. En concreto los drivers para el control de los motores eran muy voluminosos, teniendo en cuenta que se usaban 21. Ahora se usarán 3 drivers más en principio, y se pretende que el robot sea más ligero y menos voluminoso. Se necesitan drivers ligeros, pequeños, de bajo consumo y que cumplan con las especificaciones de los anteriores o las mejoren.

La información de los sensores que gestionan los microprocesadores, debe ser actualizada en tiempo real. Las consultas entre los microprocesadores debe ser lo más rápida posible. El acceso desde un medio externo al sistema, también debe ser eficaz, para posibilitar un control del robot desde Internet u otro medio físico. Se ha de estudiar las posibilidades de comunicación alámbrica e inalámbrica disponibles actualmente.

En definitiva se busca una solución para los problemas derivados de las anteriores versiones, y teniéndolos en cuenta diseñar un nuevo sistema actualizado acorde a las nuevas tecnologías disponibles.



Capítulo 3: El driver ISCM8005

Introducción del driver. Descripción, pruebas y proceso de preparado. El lenguaje TML.

3.1 INTRODUCCIÓN

En este capítulo se pretende evaluar la parte de los controladores del sistema motor, lo que se considera el entorno del control de los ejes del movimiento del robot humanoide. El elemento encargado de ello es el driver. Un driver es un actuador que se encarga de regular el flujo de intensidad que recibe el motor según las órdenes que reciba éste de la unidad central de control, para poder realizar los movimientos del robot de la forma deseada.

Necesitamos un driver de pequeñas dimensiones para poder ubicarlo cómodamente dentro del humanoide, que pueda proporcionar la suficiente potencia para alimentar los motores encargados de mover las articulaciones del humanoide, además de poseer una gran capacidad de programación. Se parte del driver utilizado en el prototipo anterior, el Harmonica A5/50 CAN de Elmo Motion Control cuyas características más importantes se reflejan en el apartado 2.2.6 de este proyecto. Por sus cualidades, hemos observado que la tarjeta ISCM8005 desarrollado por la empresa Technosoft puede ser un buen sustituto.

Figura 3.1: ISCM8005 de Technosoft

Este driver ofrece una amplia variedad de herramientas de programación. Posee un lenguaje de programación propio a nivel de microcontrolador con el que se pueden crear programas, funciones y tablas que pueden ser llamados desde el programa principal. Estos programas quedan almacenados en la memoria EEPROM del driver y en el momento de la llamada pasan a la memoria RAM. Se analizarán los modos posibles de movimiento, siempre enfocados para una programación en protocolo CANOpen. La razón de esto es que es el protocolo que se va usar en la red de comunicaciones de CANbus que une la unidad de control con cada uno de los drivers.

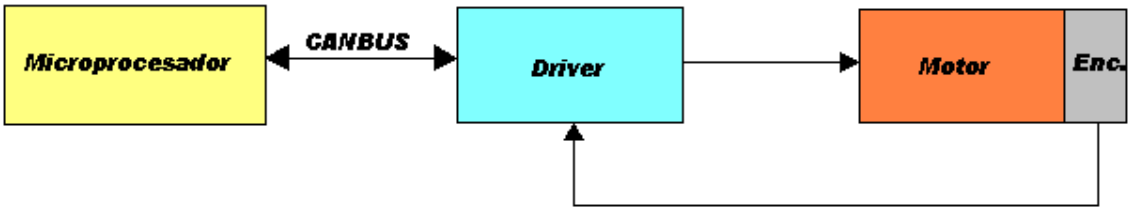


Figura 3.2: Diagrama global del driver

En este capítulo también se explicará todo el procedimiento necesario para preparar un driver, explicando las distintas conexiones a realizar entre el driver y los motores, los sensores de posición, los puertos de comunicación y la alimentación. Por otro lado también se explicará de una manera global el lenguaje usado por el driver para crear los programas y funciones que almacena en la memoria, el TML (Technosoft Motion Lenguaje).

3.2. LA TARJETA ISCM8005

3.2.1 Características

Como ya dijimos antes, el driver ISCM8005 ha sido desarrollado por la compañía Technosoft. Es un servo driver cuya funcionalidad es controlar el movimiento de los motores a su disposición. Este driver ofrece muchas ventajas frente a otros similares, las cuales podemos dividir en dos tipos: las características técnicas (tensión de alimentación, máxima corriente de continuo, máxima corriente de pico, etc.) que hacen que se adecuen a nuestras necesidades y, por otro lado, las características para el control de los motores como pueden ser los distintos modos de funcionamiento y otras utilidades que se puedan utilizar en esta tarea.

Dentro de las características técnicas podemos destacar las siguientes:

- Es un servo drive totalmente digital con inteligencia integrada y funcionalidad PLC.
- Es adecuado para motores con/sin escobillas en DC, sin escobillas AC y motores paso a paso de dos fases, los cuales se utilizarán en su mayoría para la construcción del humanoide.
- Posee conexión a puerto serie RS-232 que puede ser necesaria para un primer setup básico del driver, que necesita guardar la configuración del motor.
- Permite la conexión a puerto CAN 2.0A y 2.0B hasta 1Mbps. Esta característica es básica puesto que el humanoide tiene la necesidad de un bus de datos con una capacidad de transmisión lo suficientemente elevada como para poder gestionar todos los recursos del humanoide en tiempo real.
- Tiene entradas/salidas digitales programables y entradas analógicas que pueden ayudar a descentralizar el control. Las entradas/salidas que dispone son las siguientes:
    - Interfaz para sensores Hall lineales
    - Hasta 8 entradas/salidas digitales programables
    - 2 entradas analógicas con rangos de 0 a 5 V y +/- 10 V.
    - Cuadratura para encoder diferencial e interfaz Hall digital
- La tensión de alimentación del motor es de hasta 80V.
- Capacidad para soportar altas corrientes (5 A continuos y 16 A de pico) necesarios para ejecutar los pares de movimiento del robot.
- Está diseñado con protecciones específicas contra cortocircuitos, sobre-intensidades, sobre-tensiones y fallos de masa que ayudarán a que no se dañe el driver.

Las características de control más representativas que ofrece el driver pueden ser las siguientes:

- Posee distintos modos de control: torque, velocidad, posición; "electronic gearing", contorneo, copiador; emulador de motor paso a paso; control de variables externas (presión, flujo, temperatura etc.) de los que se hablará más adelante.
- Permite el uso de un lenguaje propio de Technosoft (TML) que aporta una serie de instrucciones para definición y ejecución de secuencias de movimiento en:
  - Control de ejes individuales o multi-ejes (modo maestro o esclavo)
  - Operaciones independientes con secuencias de movimiento almacenadas.

El uso del lenguaje TML permite entre otras cosas, crear funciones y programas que pueden ser llamadas desde la unidad de control y por ello descentralizar el control de movimientos. Además de funciones, se pueden predefinir tablas que estén cargadas en la memoria EEPROM del driver y sean ejecutadas en cualquier momento.

3.2.2 Comparativa entre Harmonica A5/50 CAN e ISCM8005

Para comprobar que la tarjeta ISCM8005 es una buena sucesora de la utilizada en los prototipos RH-0 y RH-1, es necesario hacer una comparación de sus características y ver cuál de ellas se ajusta mejor a nuestras necesidades.

Las características principales del driver Harmonica A5/50 CAN de Elmo Motion Control vienen citadas en el apartado 2.2.6 de este proyecto y las del ISCM8005 en el apartado anterior (3.2.1). La siguiente tabla resume todas estas características para poder realizar una comparativa de ambos drivers de manera más sencilla:

Característica	ISCM8005	Harmonica A5/50 CAN
Tensión de alimentación del motor [V]	12-80	50
Tensión de alimentación de la tarjeta [V]	12-48	50
Intensidad de continuo / Intensidad de pico [A]	6 / 15	5 / 7
Intensidad RMS [A]	4,2	3,5
Potencia de salida [W]	400	250
Tipos de motores que acepta	Con/sin escobillas DC Sin escobillas AC Motores paso a paso de dos fases	Con/sin escobillas DC
Tipo de realimentación que acepta	Encoder relativo Encoder diferencial Sensor Hall DC tacómetro para el bucle de control de la velocidad.	Encoder Incremental Sensores Hall Encoder Incremental con Sensores Hall para conmutación Encoder absoluto Resolver. DC tacómetro para el bucle de control de la velocidad. Potenciómetro analógico para el bucle de control de posición. Tensión de alimentación
Frecuencia máxima de realimentación	5 MHz	5 MHz
Entradas/Salidas	12 para diversos usos: - 8 salidas/entradas digitales - 2 entradas analógicas - entradas de otros usos	9 de diversos usos: -6 entradas digitales -2 salidas digitales -1 entrada analógica

Modos de control	Corriente/Par Velocidad Posición Electronic gearing Contorneo Copiador Emulador de motor paso a paso Control de variables externas (presión, flujo, temperatura etc.) Modos avanzados: (PT, PVT, ECAM)	Corriente/Par (velocidad en bucle cerrado hasta 14 KHz). Velocidad (velocidad en bucle cerrado hasta 7 KHz). Posición (velocidad en bucle cerrado hasta 3.5 KHz). Modos avanzados: PTP, PT, PVT, ECAM, Lazo Dual (con encoder auxiliar).
Puertos de comunicación	RS-232 CAN	RS-232 CAN
Capacidad de programación	Alta: Programación en TML, permite guardar programas y funciones TML y ser llamados desde CAN, captura de eventos e interrupciones	Media: Captación de eventos e interrupciones, programación a base de órdenes de movimiento en lenguaje EHL.
Dimensiones [mmxmm]	70x50	82x75
Peso [gr]	35	150

Tabla 3.1: Comparativa HARMONICA/ISCM8005

Por otro lado, se necesitan conocer los requerimientos que tiene el RH-2 para tenerlos en cuenta a la hora de hacer la elección del driver. Los aspectos como el espacio que tiene el robot para poder incluir los drivers, la tensión y potencia que necesitan los motores, etc. son totalmente influyentes y determinativos a la hora de decidirse por uno u otro.

Los motores que se van a emplear son unos de tipo DC con escobillas de la compañía Maxon de 150 W. Como se observa, ambas tarjetas cumplen el requisito de potencia y pueden suministrar la necesaria para el correcto funcionamiento.

El espacio disponible que tiene el humanoide para almacenar las baterías, los procesadores y demás es bastante reducido. Por ello una de las condiciones críticas es el tamaño del driver. Como se puede comprobar en la tabla anterior, la

tarjeta de Technosoft es sensiblemente más pequeña que su competidora. Además el peso del driver de Elmo es más de 4 veces superior al de Technosoft, por lo que esto es un punto a favor del nuevo candidato.

Otro aspecto a considerar y además de bastante importancia, son las oportunidades que ofrece cada driver a la hora de la programación. Es posible que la mayoría de las funciones no se utilicen en el primer momento de programar los movimientos del robot, pero a medida que se quiera optimizar el tiempo de los ciclos de transmisión, éstas pueden tener un valor crítico. En el caso del driver ISCM8005, se ha podido comprobar que tiene la capacidad de hacer llamadas mediante objetos del protocolo CANOpen, a funciones y programas almacenados en la memoria que hayan sido previamente cargados. Se pueden llamar hasta 10 funciones distintas y un programa. La complejidad de las funciones puede ir desde simples funciones para obtener alguna variable del driver hasta complejos comandos que hagan que se pueda liberar a la unidad de control de ciertas tareas. Esto no es así en el caso del driver HARMONICA de Elmo que no posee ninguna característica semejante que permita sacar mayor partido a posibles programas que se almacenasen en la memoria de la tarjeta.

En cambio, el driver de ELMO presenta la ventaja de poder ofrecer un doble lazo de realimentación. Esto es importante en nuestro caso ya que necesitamos conocer la posición real de un grado de libertad con un encoder absoluto y contrastarla con la posición que nos da el encoder relativo puesto en el mismo eje del motor.

En el caso de la tarjeta ISCM8005, que no tiene esta característica y además no se le pueden conectar en el lazo de realimentación encoder absolutos, este problema se solventa incorporando un circuito que controle la señal del encoder absoluto y éste la transmita a la CPU. Esto implica que se necesitaría un circuito de control por cada grado de libertad y además habría que añadir un nodo de CANbus para poder transmitir la señal a lo largo del bus de datos.

Analizando los pros y los contras de cada tarjeta y dando prioridad a las características técnicas que satisfagan las necesidades de los motores que se van a utilizar para mover el robot, así como a las posibilidades de cara al futuro que podría ofrecer el driver, se puede llegar a la conclusión de que el controlador ISCM8005 de Technosoft es el que mejor se acerca a nuestros requerimientos.

3.3 PREPARACIÓN DEL DRIVER

3.3.1 Conexionado

El driver tiene unas dimensiones de 70x50 mm, el conector general presenta 17 patillas por cara, lo que hace un total de 34 pines.

- Cara A:

En la cara A se distingue fácilmente el microprocesador y el conector JP1 (que solo se usará para precargar el firmware en caso de que éste se borre accidentalmente en un proceso rutinario). También son fácilmente distinguibles condensadores así como pequeños chips de distintas finalidades.

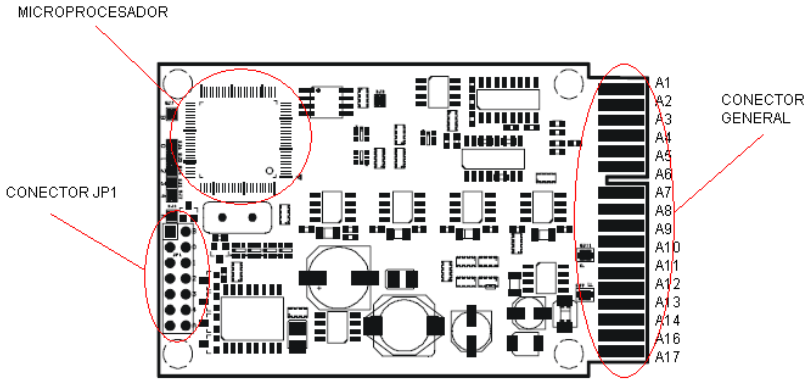


Figura 3.3: Cara 'A' del driver

- Cara B:

En esta cara se distingue el puente de transistores encargado de generar algunas de las órdenes de control de los motores.

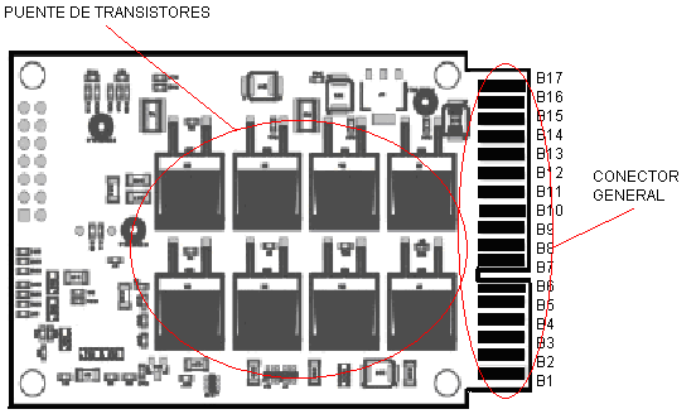


Figura 3.4: Cara 'B' del driver

En las tablas 3.2 y 3.3 se describen los pines de las caras A y B del driver, diciendo su tipo, el nombre y la función que desempeña:

Pin	Nombre del pin	Tipo	Función
A1	+Vmot	I	Terminal positivo de la alimentación del motor: de 12 a 80V
A2	A / A+	O	Fase A para motores brushless Terminal + para motores brrushed
A3	B / A-	O	Fase B para motores brushless Terminal – para motores brrushed
A4	C / B+	O	Fase C para motores brushless
A5	FRENO / B-	O	Salida para el freno
A6	+5V	O	5V generados internamente
A7	ENCA+	I	Señal A del encoder Señal A+ del encoder diferencial
A8	ENCB+	I	Señal B del encoder Señal B+ del encoder diferencial
A9	ENCZ / CAPI+	I	Señal Z del encoder Señal Z+ del encoder diferencial
A10	H1	I	Señal 1 del sensor Hall
A11	IO#38 / PULSE	I/O	Salida digital de 3.3V Entrada digital de 5V
A12	IN#2 / LSP	I	Entrada de 5V
A13	ENABLE	I	Conectar a +5V para desactivar las salidas PWM
A14	IO#13 / FDBK	I/O	Salida digital de 3.3V Entrada digital de 5V Entrada analógica de 0V a 5V. Se puede usar como realimentación de posición analógica o velocidad del tacho
A15	GND	-	Tierra
A16	CAN_H	I/O	Línea positiva del bus CAN
A17	TX232	O	Línea de transmisión del puerto RS-232

Tabla 3.2: Descripción de los pines de la cara 'A' del driver

Pin	Nombre del pin	Tipo	Función
B1	+Vmot	I	Terminal positivo de la alimentación del motor: de 12 a 80V
B2	A / A+	O	Fase A para motores brushless Terminal + para motores brrushed
B3	B / A-	O	Fase B para motores brushless Terminal – para motores brrushed
B4	C / B+	O	Fase C para motores brushless
B5	FRENO / B-	O	Salida para el freno
B6	+Vlog	I	Terminal positivo de la alimentación del driver
B7	ENCA- / LH1	I	Señal A- del encoder diferencial
B8	ENCB- / LH2	I	Señal B- del encoder diferencial
B9	ENCZ / LH3	I	Señal Z- del encoder diferencial
B10	H2	I	Señal 2 del sensor Hall
B11	H3	I	Señal 3 del sensor Hall
B12	IN#24 / LSN	I	Entrada de 5V
B13	RESET	I	Conectar a +5V para resetear la tarjeta
B14	IO#14 / REF / DIR	I/O	Salida digital de 3.3V Entrada digital de 5V Entrada analógica de 0V a 5V. Se puede usar como referencia analógica de posición, de velocidad o de torque.
B15	GND	-	Tierra
B16	CAN_H	I/O	Línea negativa del bus CAN
B17	TX232	O	Línea de recepción del puerto RS-232

Tabla 3.3: Descripción de los pines de la cara 'B' del driver

3.3.1.1 Motores

Al driver se le pueden conectar multitud de tipos de motores: brushless (sin escobillas), paso a paso de 2 y 3 fases y brushed (con escobillas). Sin embargo, para el prototipo RH-2 sólo se van a utilizar motores brushed, con la posibilidad de utilizar también brushless por lo que solo se analizarán las conexiones de estos dos tipos de motores.

- Motor Brushless:

Los motores brushless tienen 3 fases (A, B y C) que se deben conectar a los pines A2, A3 y A4 respectivamente. La masa se ha de conectar al pin A15. La siguiente figura representa el resultado final del conexionado:

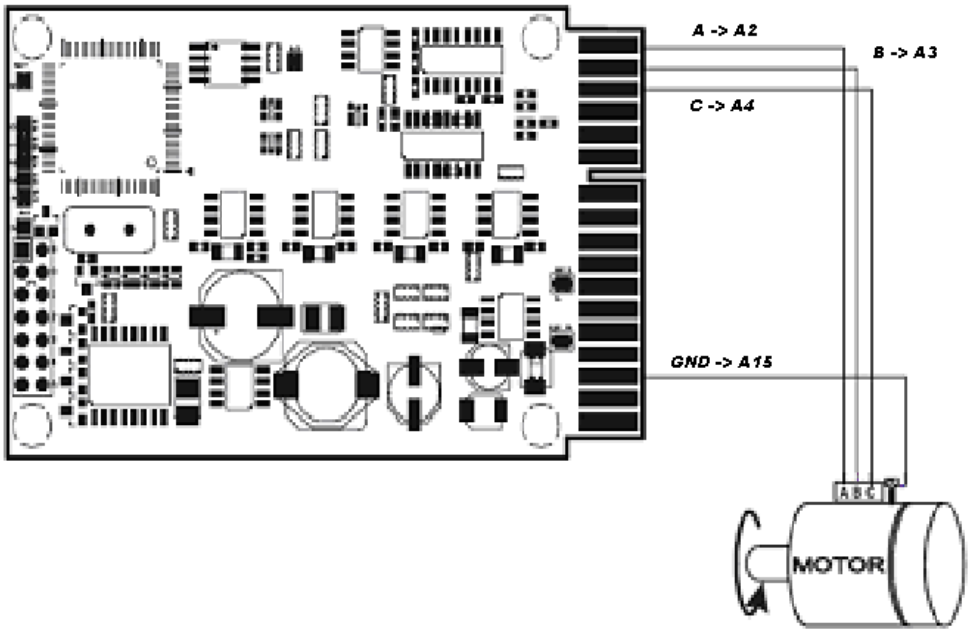


Figura 3.5: Diagrama de conexionado de un motor brushless

- Motor DC:

Los motores DC o brushed solo tienen 2 conectores de alimentación, el positivo y el negativo. El conector + se conecta al pin A2 del driver y el conector – al pin A3. La masa, de igual manera que el caso anterior, se ha de conectar al pin A15. La siguiente figura representa el esquema final de conexiones:

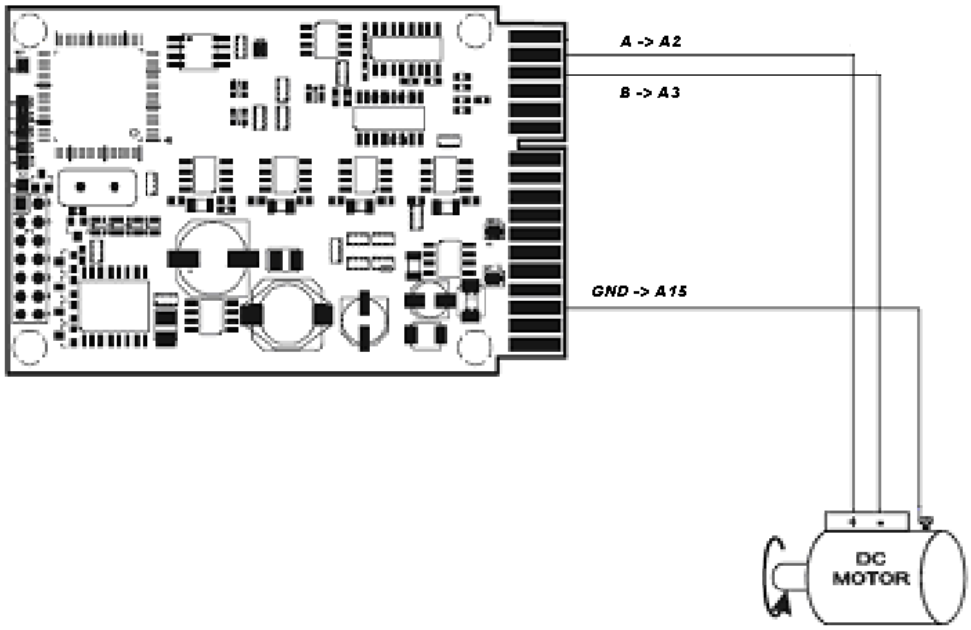


Figura 3.5: Diagrama de conexionado de un motor brushed

Como se puede apreciar, el conexionado es muy semejante, con la única diferencia que el motor sin escobillas tiene una fase más conectada al pin 4 de la cara A del driver.

3.3.1.2 Sensores de realimentación

Para el caso de la realimentación ocurre algo similar al caso de los tipos de motores. Aunque el driver permite la conexión de multitud de tipos de sensores de posición, solo se va a mostrar las conexiones del encoder y el encoder diferencial. La diferencia entre ambas conexiones estriba en que el encoder diferencial posee por cada señal digital, otra señal igual pero invertida. Las figuras son las siguientes:

- Encoder:

El encoder no diferencial presenta cinco conectores, tres de ellos generan los pulsos de posición y los otros dos son los de masa y alimentación. Normalmente los conectores que generan los pulsos reciben los nombres de ENCA+ ENCB- y ENCZ que se tienen que conectar respectivamente a los pines A7, A8 y A9 de la tarjeta. La alimentación se ha de conectar al pin A6 y la masa al pin A15. LA siguiente figura muestra el conexionado final:

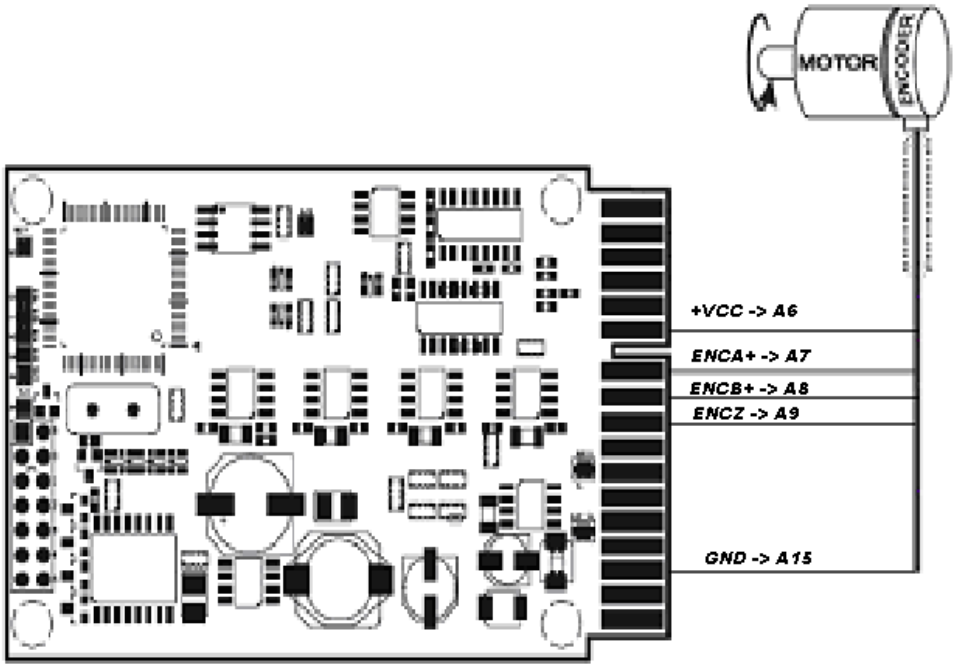


Figura 3.7: Diagrama de conexionado de un encoder

- Encoder diferencial:

Como se dijo anteriormente, el encoder diferencial tiene el doble de conectores de la señal de posición (seis), por lo que junto a los de masa y alimentación hace un total de ocho. Los de alimentación y masa se conectan como en el caso anterior en los pines A6 y A15 respectivamente. Por otro lado ahora tendremos las señales ENCA+, ENCA-, ENCB+, ENCB-, ENCZ+ y ENCZ- cuyas uniones son las siguientes:

- ENCA+ al pin A7
- ENCA- al pin B7
- ENCB+ al pin A8
- ENCB- al pin B8
- ENCZ+ al pin A9
- ENCZ- al pin B9

El conexionado final se muestra en esta figura:

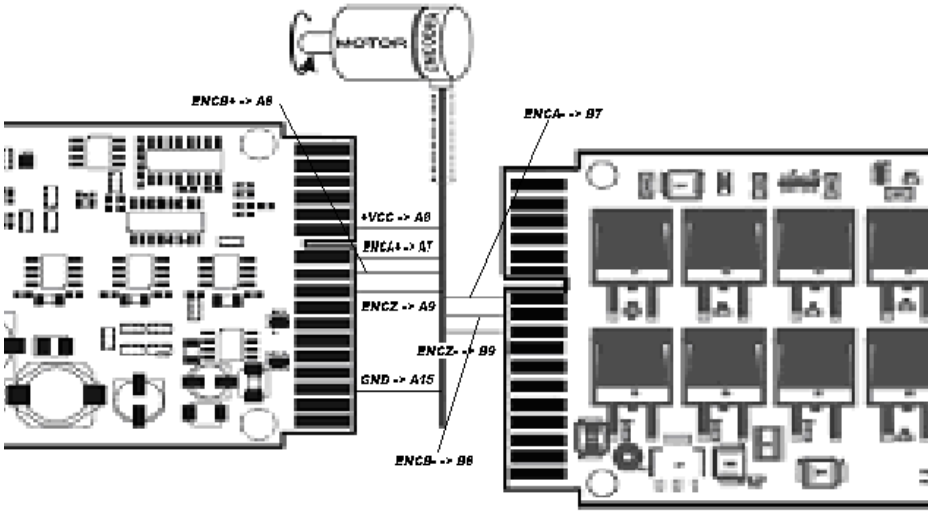


Figura 3.8: Diagrama de conexionado de un encoder diferencial

3.3.1.3 Conexión de la alimentación

En cuanto a la alimentación, el driver recibe por dos pines separados el suministro de tensión propia y la tensión de alimentación del motor. La tensión propia está comprendida en los rangos de 12 a 48 V, siendo la típica 24 V. La tensión de alimentación del motor está comprendida entre los 12 y 80 V, dependiendo ésta del motor a conectar. La tensión positiva de alimentación del driver va al pin B6 de la tarjeta y la masa al pin A15. La alimentación del motor se conecta al pin A1 y la masa al igual que en el caso anterior va dirigida al pin A15. El conexionado queda de la siguiente forma:

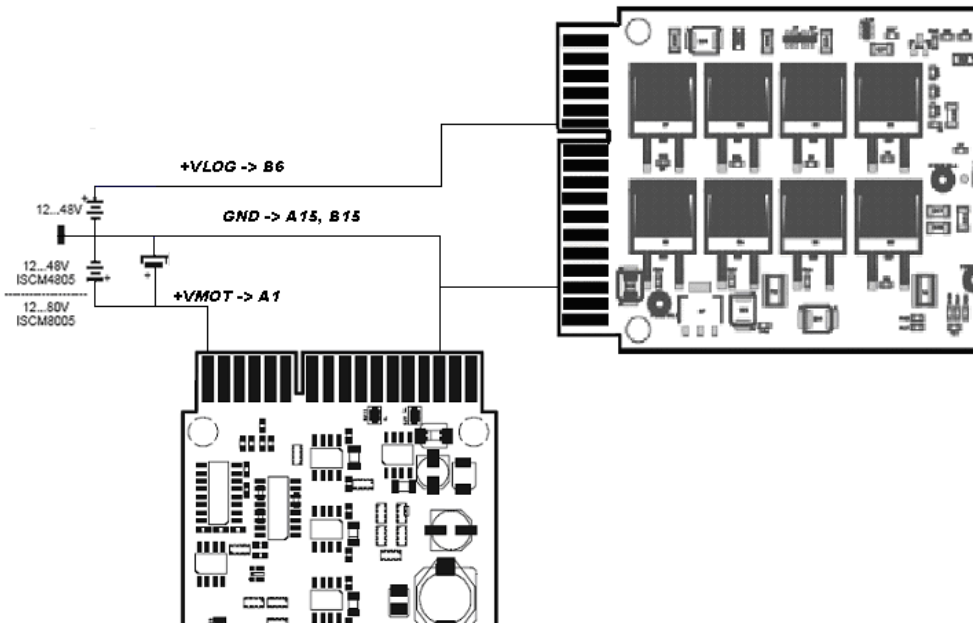


Figura 3.9: Diagrama de conexionado de la alimentación

CONDENSADOR DE ALIMENTACIÓN

Technosoft recomienda que haya un condensador en las proximidades de la alimentación del motor. El condensador debería estar posicionado a 10 cm del borde de la placa, a una distancia máxima de 20 cm. El valor recomendado para este condensador es de 100 μF o más, que ha de estar adaptado al voltaje.

También se recomienda si hay frenadas bruscas o inversiones del movimiento, que para limitar posibles sobretensiones producidas por el retorno de la energía hacia la fuente, se incluya un condensador en la alimentación del motor de mayor capacidad que el mencionado anteriormente. Al producirse está sobretensión, podría saltar la protección que posee el driver.

La capacidad de este condensador se calcula con la siguiente fórmula:

$$C \geq \frac{2 \cdot E_M}{U_{MAX}^2 - U_{NOM}^2} - C_{Driver}$$

Donde:

U_{MAX} = 95 V es el límite de la protección contra sobretensiones.

C_{Driver} = 0 μF es la capacidad interna del driver

U_{NOM} = 80 V es la tensión nominal de la alimentación del motor

E_M = es la energía total que fluye hacia la fuente en Julios. Se calcula con la siguiente fórmula:

$$E_M = \frac{1}{2}\left(J_M + J_L\right) \cdot \varpi_M + \left(m_M + m_L\right) \cdot g \cdot \left(h_{inicial} - h_{final}\right) - 3 \cdot I_M^2 \cdot R_{Ph} \cdot t_d - \frac{t_d \cdot \varpi_M}{2} \cdot T_F$$

Donde:

J_M es la inercia total del rotor [kgm^2]

J_L es la inercia total de la carga vista desde el eje después de la transmisión [kgm^2]

ϖ_M es la velocidad angular del motor después de la desaceleración [rad/s]

m_M es la masa del motor cuando se mueve en un plano no horizontal [kg]

m_L es la masa de la carga cuando se mueve en un plano no horizontal [kg]

g es la aceleración de la gravedad [m/s^2]

$h_{inicial}$ es la altitud inicial del sistema [m]

h_{final} es la altitud final del sistema [m]

I_M es la corriente del motor durante la deceleración [$A_{RMS} / fase$]

R_{Ph} es la resistencia por fase del motor [Ω]

t_d es el tiempo para decelerar [s]

T_F es el torque total de fricción visto desde el eje del motor [Nm]

El valor de J_M es conocido puesto que viene indicado en las hojas características de cada motor. Lo mismo ocurre con los valores de R_{Ph} y m_M. Sin embargo, los valores de J_L, m_L, I_M, t_d y de T_F han de ser obtenidos mediante el estudio dinámico del robot, haciendo pruebas con las articulaciones para ver las distintas inercias, variaciones de altura y cálculo de tiempos e intensidades de demanda.

Al faltar datos que se adquirirán con pruebas futuras con los motores, Technosoft recomienda que el valor de partida del condensador sea de 10000 μF.

3.3.1.4 Conexionado del CANbus

El CANbus consta de 3 cables: el terminal de tierra, el terminal CAN_H y el terminal CAN_L. Éstos se conectan al driver de la siguiente forma:

- GND a los pines A15 y B15
- CAN_H al pin A16
- CAN_L al pin B16

La siguiente figura muestra como queda el conexionado:

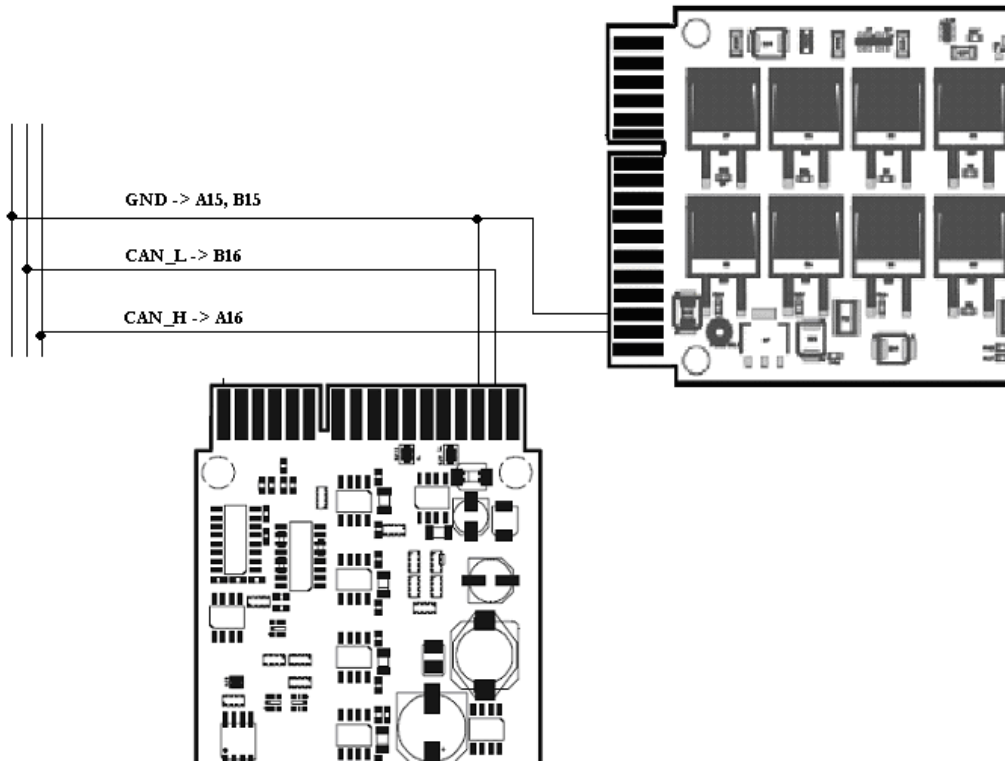


Figura 3.10: Diagrama de conexionado del CANbus

3.2.2 Cambios de firmware

Durante las experimentaciones fue necesario hacer cambios en el firmware del driver para adaptarlo al protocolo con el que iba a realizarse la comunicación. Para realizar cambios en el firmware hay que tener el driver conectado a un Pc mediante conexión por puerto serie RS-232. Una vez que se tiene conectado, hay que ejecutar el programa Firmware Programmer que viene incluido en el paquete de software del driver.

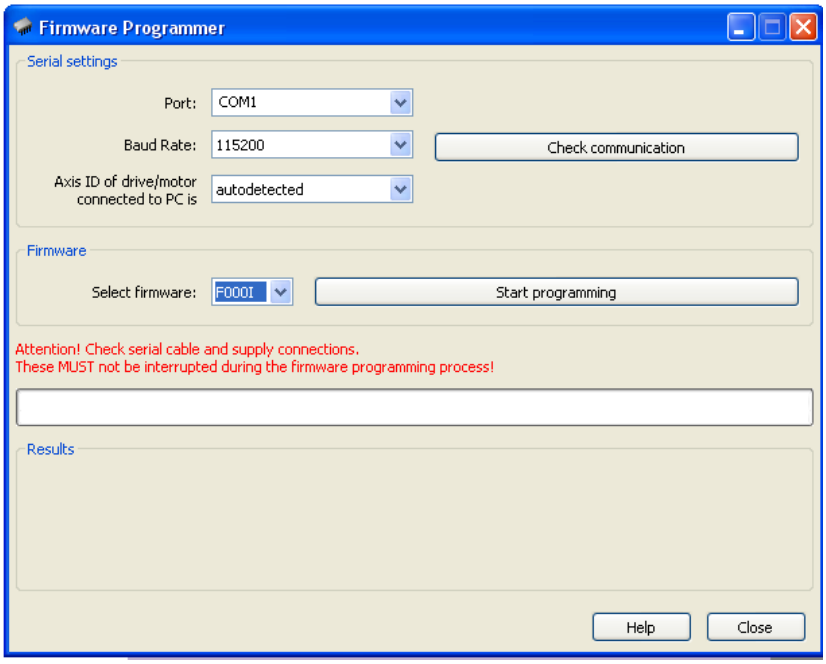


Figura 3.11: Pantalla inicial del Firmware Programmer

Una vez abierto el programa se pulsa en "check comunication" para comprobar la conexión con el driver y el firmware que lleva. La siguiente tabla muestra la versión de firmware que hay que introducir según el protocolo a utilizar:

Protocolo	RS-232	CANOpen
Firmware	F000I	F250C

Tabla 3.4: Valores del firmware según el protocolo

Es muy importante que antes de ejecutar el cambio, se compruebe que el driver está alimentado correctamente puesto que si se quedase sin tensión, el driver podría estropearse. Otra cuestión a tener en cuenta es que el PC no debe bloquearse durante el proceso, ya que la carga del firmware puede resultar errónea y de esta forma resultaría también el driver inutilizado.

Esto sucedió en una ocasión durante el desarrollo de las pruebas del proyecto. Tras comprobar que no se podía restaurar de modo alguno la configuración del driver, Technosoft nos proporcionó un pequeño programa con el que pudimos devolver el driver al estado de fábrica.

El procedimiento que hubo que seguir para restaurarlo fue el siguiente:

- El primer paso fue colocar un jumper en la posición 1-2 del conector JP1 (FU/Norm - cerrado) para habilitar la actualización del firmware.
- Después hubo que descomprimir el archivo que nos enviaron y ejecutarlo previa conexión del driver al Pc mediante puerto serie RS-232. Al ser ejecutado se nos pidió una contraseña que venía adjunta con la información remitida por Technosoft.
- Cuando el programa muestre un mensaje pidiendo cambiar el interruptor, simplemente hay que cambiar el jumper del JP1 a su posición inicial y pinchar en ok. Se resetea el driver y ya tendrá cargado el firmware estándar (la versión F000I).

Si se quisiese cambiar el firmware simplemente habría que realizar el procedimiento normal con el programa Firmware Programmer.

3.3.3 Procedimiento para configurar el driver para un nuevo motor

Cada vez que se conecta un motor al driver, ha de obtener la información sobre el tipo de motor que tiene conectado, sus intensidades de funcionamiento, encoders utilizados y otros parámetros necesarios para que funcione correctamente. Para ello se utiliza del programa EasyMotion Studio la funcionalidad para poder configurar los datos del motor. Una vez abierto o creado un nuevo proyecto, hay que ir a la ventana de proyect (si no se ve, dar a View -> Proyect)

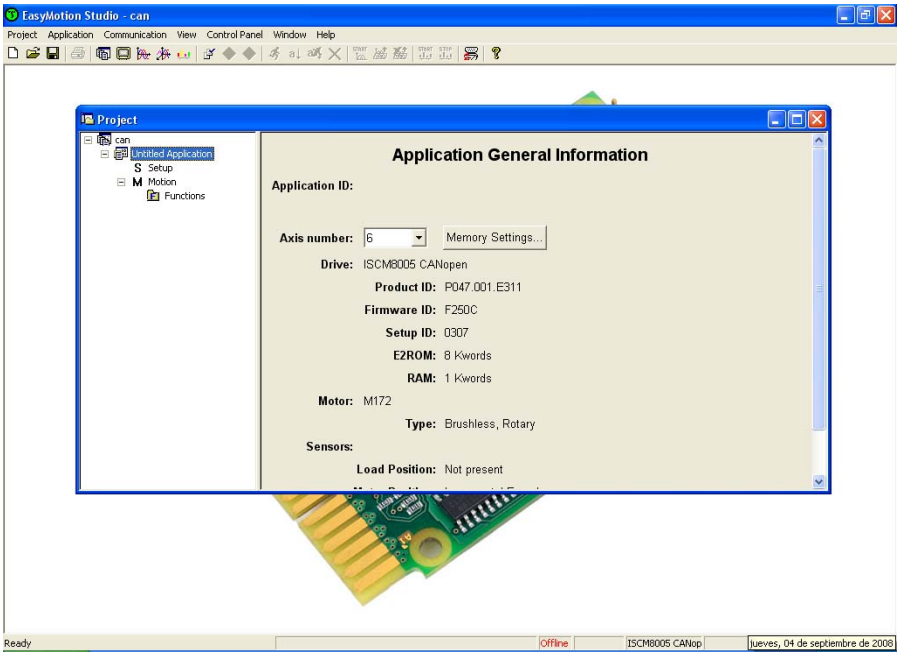


Figura 3.12: Pantalla inicial del proyecto

Una vez en ella, en el menú de la izquierda, dar a Setup. Aparecerán una serie de opciones desde las cuales se podrá crear una nueva configuración del driver, cargar una ya creada, descargarla al driver, etc.

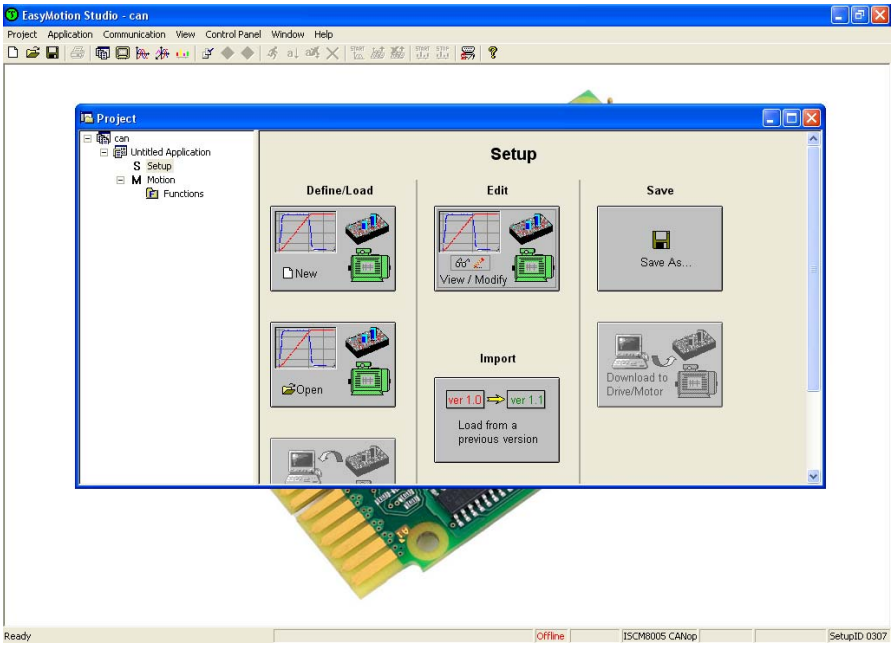


Figura 3.13: Pantalla del setup

Para crear una nueva configuración hay que pulsar en Edit, tras lo cual aparecerán dos nuevas ventanas, una para configurar los parámetros del motor y otra para los de control del driver:

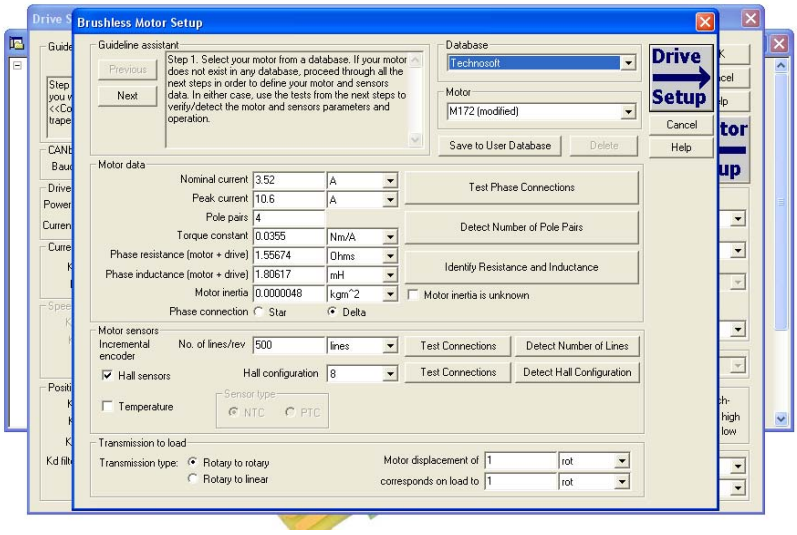


Figura 3.14: Pantalla de setup del motor

Una de ellas es para configurar el driver y otra se usará para poner los parámetros del motor. Hay que mencionar que según el tipo de motor que se haya elegido al crear el proyecto, algunas de las opciones de configuración serán distintas o bien pueden estar deshabilitadas. El primer paso es comprobar que nuestro motor está en la base de datos que tiene el programa. En Database se selecciona el fabricante y en Motor el modelo. Si nuestro motor no se encuentra en dicha base habrá que seleccionar User y configurar todos los parámetros requeridos.

Los parámetros a editar son:

- Intensidad nominal
- Intensidad de pico
- Inercia del motor
- Constante de par
- Resistencia por fase
- Inductancia por fase
- Pares de polos (según el tipo de motor a configurar)

Todos estos datos han de venir en la hoja de características del motor que tenga el distribuidor o fabricante.

Debajo de los parámetros del motor, se configuran las características del encoder, sensor hall, etc. que estén conectados al driver. Los datos a incluir (normalmente, ya que las opciones disponibles dependen de los sensores conectados y del tipo de motor) son las líneas por revolución que ofrece el encoder y si hay un sensor hall conectado. Si se tiene un tacho conectado habrá que definir la ganancia que tiene este.

En esta misma ventana se pueden ejecutar distintos test para probar que las conexiones motor-driver y encoder-driver son correctas. Que las opciones de test estén disponibles o no, viene determinado por el tipo de motor y sensores que se utilicen. Si el motor es brushless se podrán realizar test para comprobar las conexiones de cada fase, otro test con el que se puede comprobar el número de polos que posee y un último para identificar automáticamente los parámetros de resistencia e inductancia por fase del motor. En caso de que el motor fuese de tipo brushed, las opciones anteriores no estarán disponibles.

Los test disponibles para el encoder son: comprobar las conexiones (con este test y girando el eje del motor se ve en pantalla como varía la orientación del eje) y detectar el número de líneas por revolución que posee. Este último test solo está disponible para algunas configuraciones y se realiza de manera automática una vez dado a "start". Si hubiese un sensor hall, con los test disponibles se puede comprobar que se ha conectado correctamente al driver y su configuración.

Una vez en la ventana de configuración del driver hay que seleccionar el tipo de control que se desea realizar sobre el motor (posición, velocidad o par). Si se pulsa en advanced, se puede seleccionar si se cierra o no distintos lazos de realimentación según el método de control. Por ejemplo si se ha escogido el de posición, nos dan la opción de cerrar el lazo de velocidad. En esta ventana también se configura la frecuencia de la señal PWM, y con ella el periodo de los lazos de control. Existen dos divisores que se pueden utilizar para variar los periodos según se quiera una realimentación rápida o lenta.

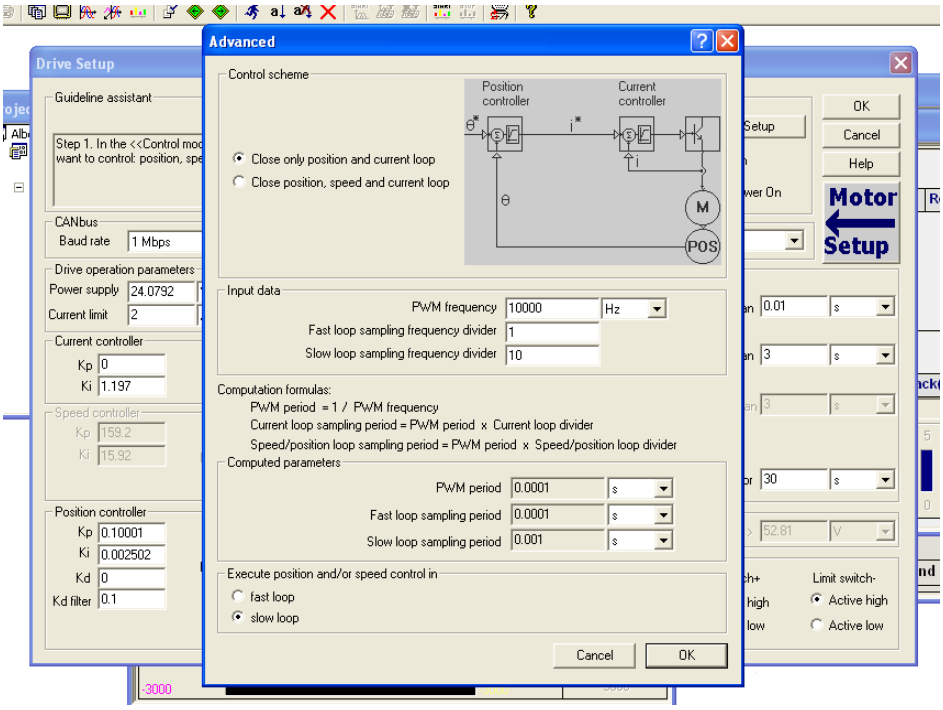


Figura 3.15: Pantalla de Advanced de los modos de control

Dependiendo del control seleccionado, algunas opciones de configuración quedarán habilitadas o deshabilitadas. Posteriormente hay que seleccionar la velocidad de transmisión del CANbus aunque supondremos que se quiere realizar a velocidad máxima y por ello se elegirá 1Mbps. El siguiente paso es verificar si el driver adquiere la posición, velocidad o par de referencia de un dispositivo externo. Después se puede ejecutar un test que comprueba la tensión de alimentación del driver.

Posteriormente se configuran los parámetros (kp, ki y kd) que utiliza el driver para controlar la corriente, posición y la velocidad. El driver tiene una opción en cada apartado para hallar cada uno de los parámetros de forma automática realizando un auto-tunning. A estos test se accede pulsando en Tune & Test apareciendo unos menús para configurar los test a realizar.

Para calcular los parámetros de la intensidad, se debe configurar la intensidad a la que se va a realizar el test según sea su nominal. Este test se ha de

ejecutar con el eje del motor bloqueado. Una vez pulsado al botón start y que haya acabado el test hay que pulsar en Tune para que el programa EasyMotion Studio automáticamente cambie los parámetros.

En el test para hallar los parámetros de control de la velocidad se puede editar el perfil que va a realizar el driver, cambiando velocidades a alcanzar, tiempos en alcanzar las velocidades, etc. Se accede a la pestaña de test y se pulsa en start para iniciar la prueba. Para que el programa ajuste automáticamente los parámetros se pulsa en Tune.

El test de los parámetros de la posición es similar al realizado en el de la velocidad. Se edita el perfil de posición que va a realizar el driver, cambiando posiciones de pico, tiempos de subida y de bajada, etc. Se accede a la pestaña de test y se pulsa en start para iniciar la prueba. Para que el programa ajuste automáticamente los parámetros se pulsa en Tune.

En la pantalla del Setup del driver además se pueden configurar las protecciones del driver frente a sobreintensidades, errores en la posición o en la velocidad. También existe la posibilidad de seleccionar el ID que queremos proporcionar al eje, o bien poner que éste quede determinado por hardware. El ID que le proporcionamos al motor es fundamental para luego realizar correctamente las comunicaciones maestro-driver. Este ID es el que utiliza el maestro para referirse a cada nodo conectado al bus de CAN.

Otros parámetros que se pueden configurar en esta ventana son el método de conmutación que usa el driver (sinusoidal o trapezoidal), la polaridad de las entradas, el modo de inicio y si hay conectado o no un freno.

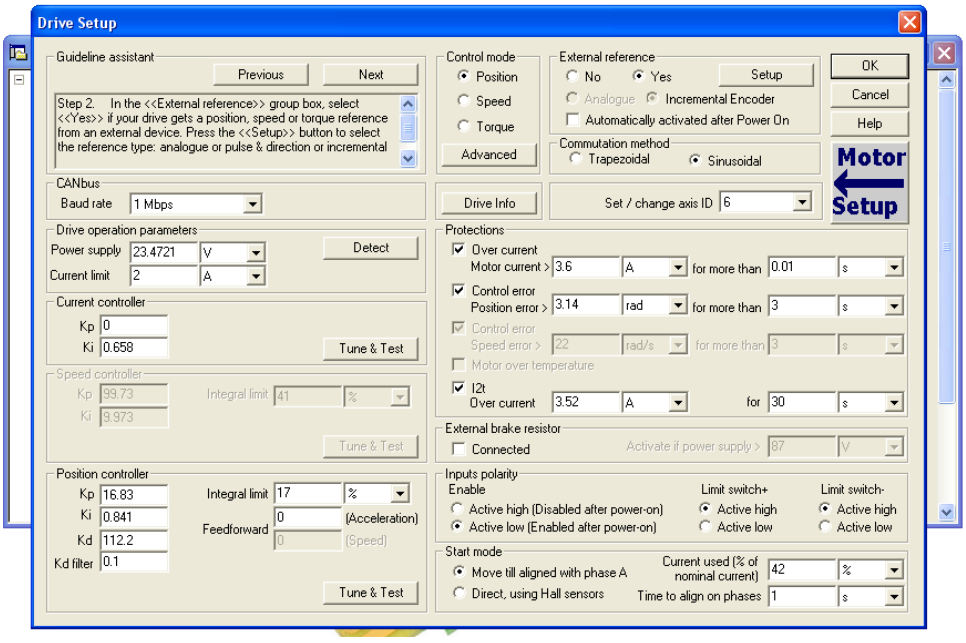


Figura 3.16: Pantalla de setup del driver

Una vez terminada la configuración se pulsa en OK y para descargar la información al driver se pulsa en el botón Download Setup to Drive/Motor de la barra de herramientas global de EasyMotion.

3.3.4 Pruebas y resultados

Los resultados más importantes a mostrar son cómo se adecua el driver a tipos distintos de motores. En nuestro caso se han utilizado un motor brushless y un motor brushed.

El motor brushless es el que incorporaba el demostrativo del driver. El fabricante del motor es Technosoft. Este motor tiene las siguientes características:

- Intensidad nominal: 3.52 A
- Intensidad de pico: 10.6 A
- Encoder incremental y sensor hall
- Pares de polos: 4

El fabricante del motor brushed es Faulhaber. Pertenece a la serie 3257 CR. Las características más importantes del motor son:

- Intensidad nominal: 1.17 A
- Intensidad de pico: 2.3 A
- Encoder incremental

Como prácticamente la mayoría de los parámetros de los motores se introduce de forma manual, interesa estudiar cómo realiza el driver el proceso de auto-tunning del motor, es decir, de que manera calcula el driver los parámetros de control de posición, velocidad e intensidad. El procedimiento para hacer este test ha sido explicado en el apartado anterior. Tras realizar el test del control de intensidad, las gráficas obtenidas se muestran en la figura 3.16, correspondiéndose la imagen superior con la del motor de Technsoft y la inferior para el motor Faulhaber. Los datos del test, con las intensidades utilizadas y los valores de constantes obtenidos se muestran en la tabla 3.5.

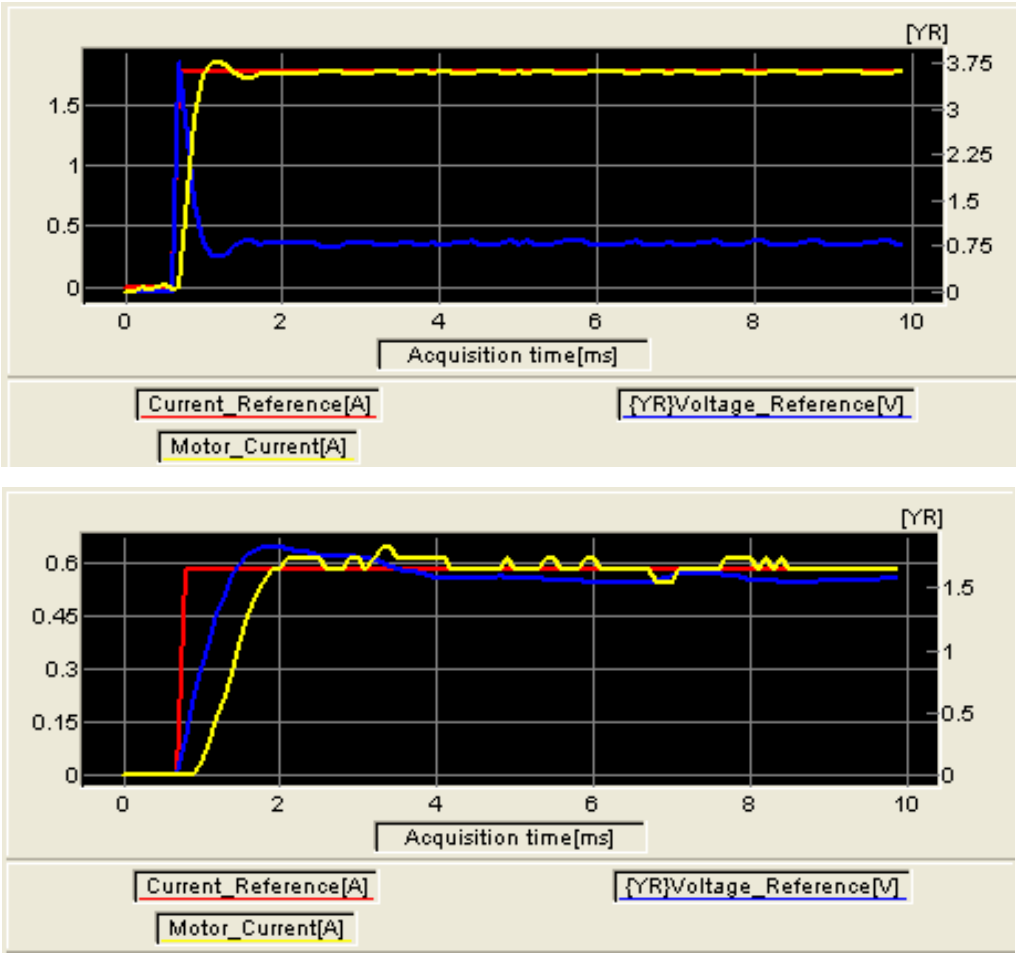


Figura 3.17: Prueba control de intensidad motor Technosoft y Faulhaber

Motor	Intensidad de prueba	Intensidad de protección	Kp	Ki
Technosoft	1.76 A	3.6 A	2.417	0.3081
Falhauber	0.585A	2.3 A	0	0.342

Tabla 3.5: Valores de intensidad y constantes obtenidas del test

A continuación se ejecuta el test del controlador de posición para que al igual que en el test de la intensidad, se obtengan los parámetros de control de lazo cerrado y asi el driver maneje de forma correcta la señal que genera el encoder y en consecuencia se minimice el error cometido. En la tabla 3.6 se recogen las gráficas que genera el test poniendo dos tipos de entrada (rampa y escalón) y la tabla 3.7 contiene los datos usados en cada test así como los valores de los controladores kp, ki y kd obtenidos para cada motor.

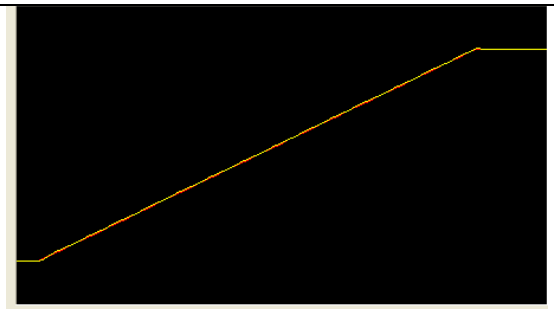
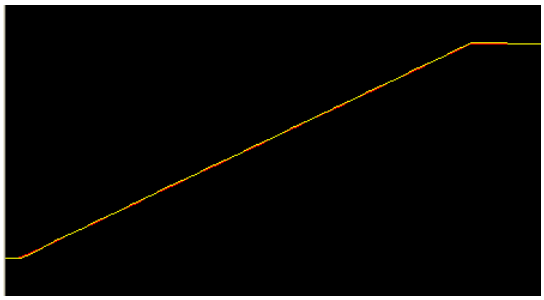
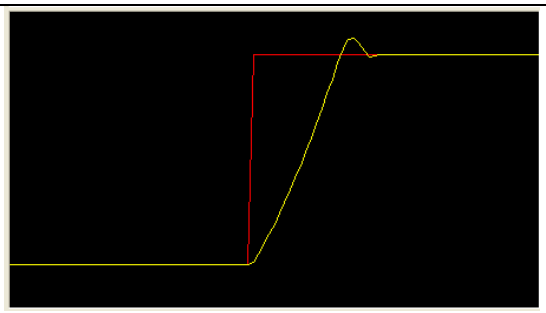
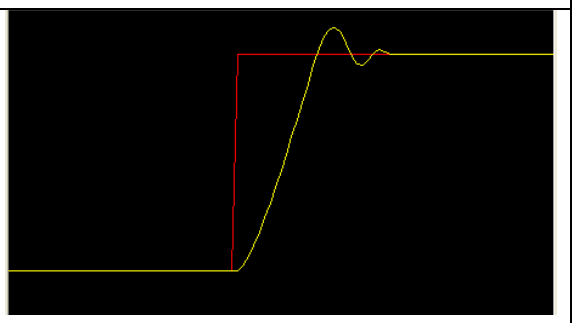
	Motor brushless	Motor brushed
Prueba rampa		
Prueba escalón		

Tabla 3.6: Gráficas del control de posición

Motor	Intensidad de prueba	Intensidad de inicio	Intensidad de protección	Tiempo de subida/bajada	Posición final	Tiempo de pico	Kp	Ki	Kd
Technosoft	2 A	50%	3.6 A	1 s*	10 rot	1s	65.02	3.251	433.5
Falhauber	2 A	-	2.3 A	1s*	10 rot	1s	0.10001	0.002502	0

Tabla 3.7: Datos de test del controlador posición

*Para el test de un perfil cuadrado, el tiempo de subida/bajada es 0.

Ya por último se realiza el test del controlador de velocidad para obtener los parámetros de control de realimentación de la velocidad. La tabla 3.8 recoge las gráficas obtenidas para cada motor y la tabla 3.9 recoge los parámetros de control.

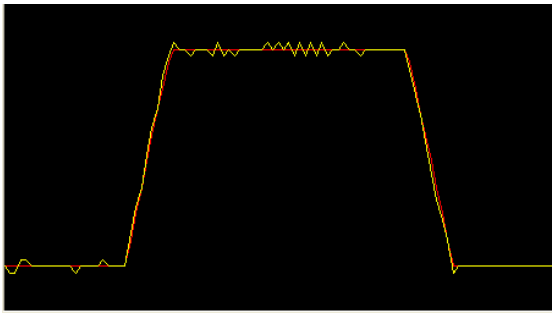
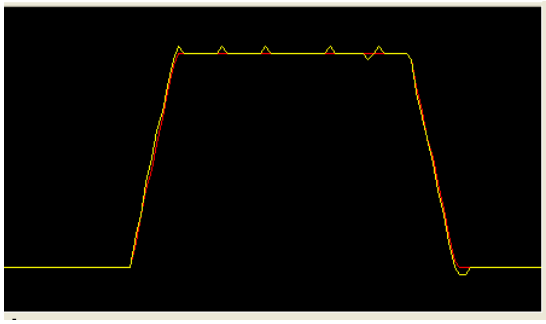
	Motor brushless	Motor brushed
Control velocidad		

Tabla 3.8: Gráficas del control de velocidad

Motor	Intensidad límite	Intensidad de protección	Velocidad final	Tiempo de subida/bajada	Tiempo de pico	Kp	Ki
Technosoft	2 A	3.6	1000 rpm	0.1 s	0.5s	399.1	39.91
Falhauber	2 A	2.3 A	1000 rpm	0.1 s	0.5s	273.9	27.39

Tabla 3.9: Datos de test del controlador posición

Como se puede observar en la tabla 3.6, el driver ante entrada escalón realiza un control subamortiguado, por lo que existen picos de sobreoscilación que pueden ser dañinos para las articulaciones del humanoide e incluso para los motores. Esto se debe a que existen cambios bruscos de aceleración/deceleración pues hay cambios en el sentido de giro del motor en los picos. El problema anterior es un fallo que tiene la funcionalidad de "autotuning" del programa Easymotion porque debería estar preparado por defecto para generar un control subamortiguado. Por ello, el cálculo de los valores de los parámetros kp, ki y kd se ha de realizar de forma teórica, pero dichos cálculos quedan fuera del objetivo de este proyecto.

3.4 MODOS DE OPERACIÓN

En este apartado se pretende describir las capacidades que posee el driver, es decir, desde los modos de control del movimiento hasta la posibilidad reaccionar a eventos o interrupciones generadas por valores determinados en las entradas analógicas/digitales o por otros sucesos.

3.4.1. Modos de control del movimiento del motor

El control del motor se puede realizar según el modo en que se encuentre el driver (control de posición, control de velocidad, etc.). Dentro de cada modo existen distintos submodos de funcionamiento.

1) Modo de control de posición.

En el modo de control de posición se puede gestionar la velocidad a la que se alcanza la posición objetivo. Pasaremos a analizar las alternativas existentes:

- Perfil trapezoidal:

El generador de referencia calcula el perfil de posición con una forma de velocidad trapezoidal, con aceleración limitada. En el modo relativo, la posición a alcanzar puede ser calculada con dos procedimientos: estándar (modo predefinido) o aditivo. En el modo relativo, la posición a alcanzar es calculada añadiendo el incremento de posición a la posición instantánea en el momento en el que la orden está siendo ejecutada. En el modo aditivo relativo, la posición a alcanzar es calculada añadiendo el incremento de posición a la posición anterior objetivo, independientemente del momento en el que la orden sea emitida.

- Perfil de curva:

El generador de referencia calcula un perfil de posición con la velocidad en forma de curva, también llamado en S. Esta forma se debe a la limitación de tirón, pues se lleva a cabo un perfil trapezoidal o triangular para la aceleración y un perfil curvado para la velocidad.

2) Modo de posición interpolado.

Para llevar el control de uno o varios ejes de forma que el control de posición esté sincronizado, se puede usar el modo de posición interpolado. Este modo puede usar el mecanismo de sincronización de tiempo para que los driver

seleccionados estén coordinados en el tiempo, basado en la sincronización y en los mensajes de tiempo de alta resolución.

El modo de posición interpolado permite a un maestro transmitir una serie de datos de interpolación a un driver. Los datos de interpolación son enviados de forma más efectiva en paquetes ya que el driver posee un buffer de entrada. El tamaño del buffer es el número de los datos de interpolación que pueden ser enviados al driver para llenar el buffer de entrada. Dentro de este modo se contemplan varias alternativas:

- Modo PT: interpola posición y tiempo de forma lineal.
- Modo PVT: interpola posición, velocidad y tiempo de forma cúbica.

3) Modo de control de velocidad

En este modo de funcionamiento, el motor queda controlado mediante la velocidad. El generador de trayectorias crea un perfil de velocidad en forma trapezoidal, con una aceleración determinada. El motor continúa a esa velocidad hasta que recibe una orden de paro o un cambio de velocidad.

4) Modo ECAM (Electronic Camming Position).

Otra alternativa para controlar la posición del motor, es mediante las denominadas tablas CAM; formadas por una serie de puntos (X,Y) donde "X" es la entrada de la tabla CAM p. ej. la posición del maestro electrónico de ECM (electronic camming master), e "Y" es la salida de la tabla CAM; por ejemplo la posición de esclavo correspondiente. Entre los puntos el driver realiza una interpolación lineal.

El esclavo puede conseguir la información de posición del maestro de dos modos:

- Vía canal de comunicación CANbus, de otro driver definido como maestro. La posición es enviada usando TechnoCAN.
- Vía una referencia externa digital de tipo pulso y dirección o de un encoder de cuadratura. Ambas opciones tienen entradas para ello. Un indexer por lo general proporciona las señales de pulso y dirección y debe ser conectado a las entradas de pulso y dirección del driver. Las señales del encoder de cuadratura son proporcionadas normalmente por un encoder ubicado en el maestro y conectado a las entradas del segundo encoder.

El tipo de referencia, p. ej. la selección entre la referencia online recibida vía el canal de comunicación y la referencia digital leída de las entradas es hecho con el objeto "tipo de referencia externo" (External Reference Type).

El modo de posición electrónico camming puede ser relativo o absoluto. En el modo relativo la salida de la tabla CAM es añadida a la posición actual del esclavo. En el modo absoluto, la salida de la tabla CAM "Y" es la posición de a alcanzar. Normalmente, las tablas CAM se cargan en la memoria EEPROM del driver con el EasyMotion o con el maestro de CANOpen y posteriormente se pasan a la memoria RAM.

5) Modos de control avanzado.

Hay formas de control más avanzadas en las que se pueden definir funciones en cada driver por medio del EasyMotion y posteriormente ser llamadas por el maestro del CAN bus. También se pueden ejecutar programas de TML creados en Easy Motion y cargar tablas CAM ya definidas.

6) Modo EGEAR (Electronic Gearing Position).

En este modo de funcionamiento un driver esclavo sigue la posición de un maestro con una relación de multiplicación programable. La forma en que el driver adquiere la información es la misma que para el caso anterior.

7) Modo de posición por referencia externa.

En esta forma de funcionamiento, el driver realiza el control de posición mediante la posición leída de la referencia externa proporcionada por otro dispositivo. Hay 3 tipos de referencia externa:

- Leída de una de las entradas analógicas del driver.
- Digital calculada por el driver de 2 formas:
  - Mediante pulsos y señales de dirección.
  - Mediante señales de cuadratura provenientes de un encoder incremental.
- De forma online recibida a través del CANbus procedente del maestro

3.4.2 Entradas/Salidas

La tarjeta ISCM 8005 posee 12 entradas/salidas que se pueden dividir en los siguientes grupos:

- 8 entradas/salidas digitales de uso general. De ellas, 4 están predefinidas como entradas y las otras 4 como salidas.
- Posee también 2 entradas para detectar transiciones de nivel alto a bajo o viceversa, que guardan la posición actual del motor cuando ocurre la transición.
- Las 2 entradas restantes se denominan entradas interruptores por límites. Estas entradas paran el motor cuando intenta moverse en una dirección determinada entrando en una zona protegida, pero permiten moverse al motor en la dirección contraria.

3.4.3 Eventos

Se pueden definir eventos o condiciones para ser monitorizados y que con su suceso se realicen una de las siguientes acciones:

- Se cambien el modo de control del motor y/o los parámetros de control.
- Se detenga el movimiento del motor con uno de los modos de parada.
- No ocurra nada

Estos eventos se pueden clasificar en 7 tipos:

1) Evento de fin de movimiento. Este evento ocurre cuando el motor alcanza la posición final programada, o bien hay una parada del movimiento.
2) Evento dependiente de la posición del motor. Se puede programar un evento que ocurra cuando la posición actual del motor sea igual o mayor/menor que una predefinida.
3) Evento dependiente de la velocidad del motor. El evento ocurre cuando la posición actual del motor es igual o mayor/menor que una predefinida.
4) Evento de tiempo de espera. Se puede programar un contador de tiempo relativo/absoluto para que una vez alcanzado un valor programado, suceda el evento.
5) Evento en función de la referencia. En este evento se compara una variable con un valor predefinido. Esta variable varía según el modo de control del driver, siendo la referencia de posición en el modo de control de posición, la velocidad en el modo de control por velocidad, etc.
6) Eventos en función del estado de entradas. Se pueden definir eventos con el estado de las siguientes entradas:

- Entradas de captura. El driver posee dos entradas de captura para detectar transiciones de una señal digital de nivel bajo a alto o de nivel alto a bajo. Cuando ocurre la transición se guarda la posición actual del motor en una variable.
- Entradas de interruptor por límite. El driver cuenta con dos entradas de este tipo, una para cada dirección. Sirven para detectar, por ejemplo, movimientos accidentales fuera de la zona de trabajo. Al igual que ocurría con las entradas de captura, este evento puede ser configurado para detectar transiciones en el nivel de la señal de bajo a alto o de alto a bajo, guardando la posición actual del motor en una variable.
- Entradas digitales de uso general. Se pueden editar eventos para que se activen cuando una de estas entradas se ponga a nivel alto o bajo después de una transición.

7) Evento en función del valor de una variable. Se compara el valor de una variable para activar un evento.

3.4.4 Interrupciones

Existe la posibilidad de definir hasta 12 condiciones que generen interrupciones que pueden ser supervisadas a la vez. A diferencia del caso de los eventos donde se esperan sucesos programados, la meta de las interrupciones es proveer una forma de reaccionar ante sucesos imprevistos. Una vez que se produce una interrupción hay un salto a la rutina de atención a la interrupción y una vez terminada de ejecutar se retorna al programa principal.

3.4.5 Homing

Los driver de Technosoft incluyen una función para encontrar su posición absoluta si no se dispone de sensores hall o encoders absolutos. Esta función se llama homing y hay cuatro fuentes disponibles para ubicar la señal homing: las entradas de interruptores por límite (negativa y positiva), la señal de interruptor de posición de inicio y un pulso índice.

Dentro del homing se nos ofrece cerca de 30 métodos distintos para encontrar la referencia absoluta combinando las 4 señales mencionadas anteriormente. Para obtener más información sobre los métodos de homing ver manual de usuario del driver.

3.5 MODOS DE FUNCIONAMIENTO. MODOS PT Y PVT

Aunque en el apartado 3.4.1 se ha descrito cada uno de los modos de control, este apartado pretende resumir los resultados obtenidos tras la experimentación con los distintos modos que posee el driver con el programa EasyMotion Studio. Después se enfoca a los modos PT y PVT, ya que son los que con mayor probabilidad se usen en el control del humanoide, para analizar similitudes y diferencias entre ambos modos. Como se dijo, los modos que posee el driver son los siguientes:

1) Modo de perfil de posición. Dentro de este modo, hay 2 sub-modos: perfil de velocidad trapezoidal y perfil de velocidad en S.
2) Modo de control de velocidad. En este modo solo se controla la velocidad a la que se mueve el motor. No se tiene control alguno sobre la posición, por lo que el motor se mueve a la velocidad indicada hasta que recibe una orden de parada o de variación de velocidad.
3) Modo de posición interpolada. Dentro de este modo, también hay 2 sub-modos: los modos PT (posición-tiempo) y PVT (posición-velocidad-tiempo). Como ya se dijo en otros capítulos, en el modo PT se hace una interpolación lineal entre puntos, mientras que en el modo PVT se hace una interpolación de tercer orden.

A continuación su muestran ejemplos de cada uno de los tres modos de funcionamiento:

Ejemplo perfil trapezoidal y ejemplo de perfil en curva

Se ejecuta un perfil trapezoidal relativo en el que el objetivo era llegar primero a 50 rotaciones con una velocidad de 2000 rpm y una tasa de aceleración de 1000rad/s^2 y luego alcanzar 70 rotaciones con una velocidad de 3000 rpm y la misma tasa de aceleración.

Se ejecuta un perfil en curva relativo en el que el objetivo era llegar primero a 50 rotaciones con una velocidad de 2000 rpm y una tasa de aceleración de 1000rad/s^2 y luego alcanzar 70 rotaciones con una velocidad de 3000 rpm y la misma tasa de aceleración. El jerk en ambos es de 6000 rad/s^3

La tabla 3.10 muestra las gráficas de los perfiles de posición y de velocidad obtenidas en ambos ejemplos, para poderse comparar con mayor facilidad y así contrastar las diferencias ya que son métodos de control bastante similares.

Como se puede observar, la diferencia radica en los cambios de velocidad, que en el perfil trapezoidal genera picos mientras en el perfil curvado el cambio se realiza de una forma más suave. Cómo es de suponer, si se utilizase uno de estos métodos de control para las articulaciones del humanoide, el mejor sería el curvado puesto que hace que los movimientos sean un poco menos bruscos, si bien tampoco se notaría demasiada diferencia a menos que el control se realizase a base de muchos trapecios siendo estos muy marcados.

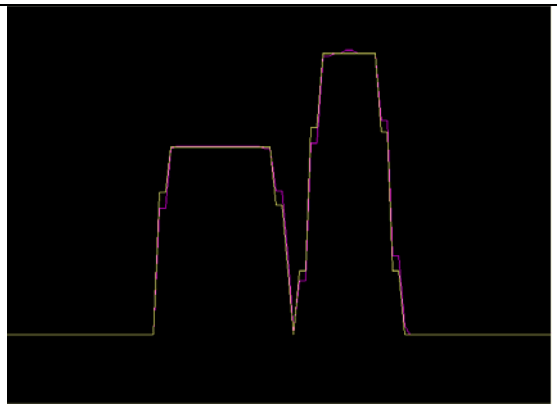
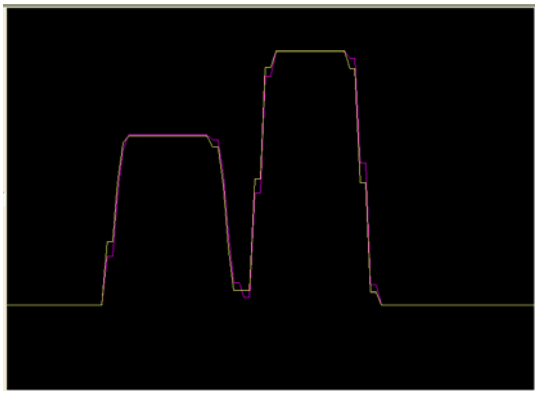
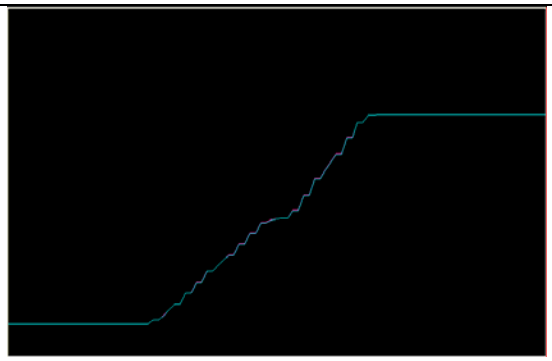
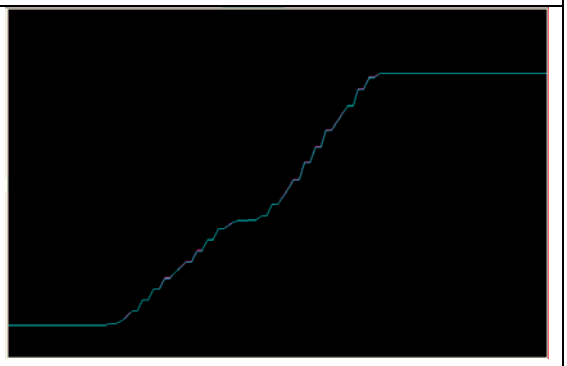
	Trapezoidal	Curva en S
Velocidad		
Posición		

Tabla 3.10: Perfiles generador por el control trapezoidal y en curva

Ejemplo de control por velocidad

Se ejecuta en modo de control por velocidad un perfil en el que se pone una velocidad de 2000 rpm y una aceleración de 100 rad/s^2.

La tabla 3.11 recoge los perfiles de velocidad y posición generados por el control de velocidad. Cómo se puede ver, la velocidad en este modo se mantiene constante todo el rato pues no tiene marcada una posición final a alcanzar. Sin embargo, podría ejecutar perfiles semejantes al control de posición trapezoidal realizando los cambios de velocidad en el mismo instante que el trapecio llega a la posición final.

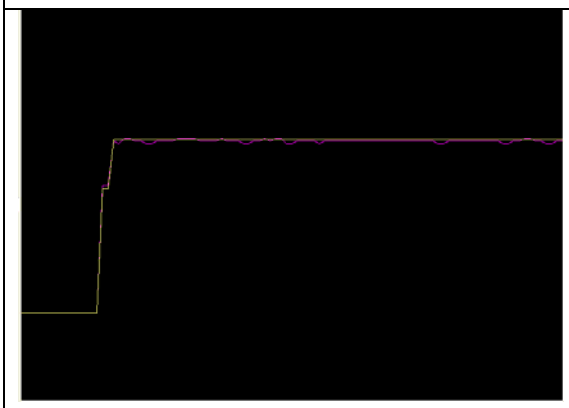
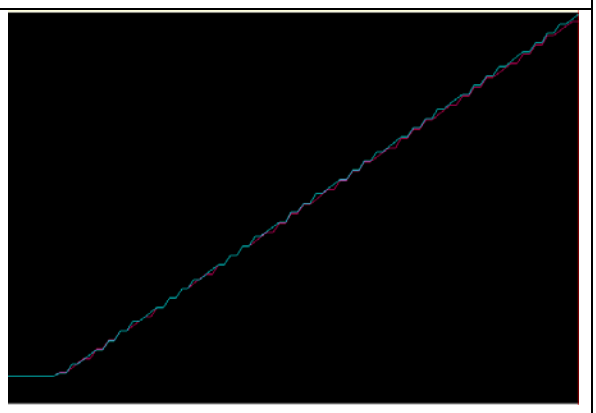
Velocidad	Posición
	

Tabla 3.11: Perfiles generados por el control de velocidad

Ejemplo modo PT y modo PVT

Para poder contrastar ambos modos, se van a mostrar tres ejemplos: uno del modo PT y dos del modo PVT, puesto que distintos valores de velocidad para unos mismos puntos (posición y tiempo) generan perfiles bastante distintos y que merece la pena comparar. En los tres ejemplos se van a cargar los mismos valores para los parámetros de posición y tiempo. En la tabla 3.12 se muestran los datos cargados en el modo PT, en la tabla 3.13 los datos del primer ejemplo PVT y en la tabla 3.14 los del segundo ejemplo PVT.

Punto	1	2	3	4	5
Posición [rot]	7.5	22.5	47.5	60	65
Tiempo [s]	0.5	1	1.5	2	2.5
Velocidad (posición/tiempo) [rpm]	900	1800	3000	1500	600

Tabla 3.12: Datos del perfil PT

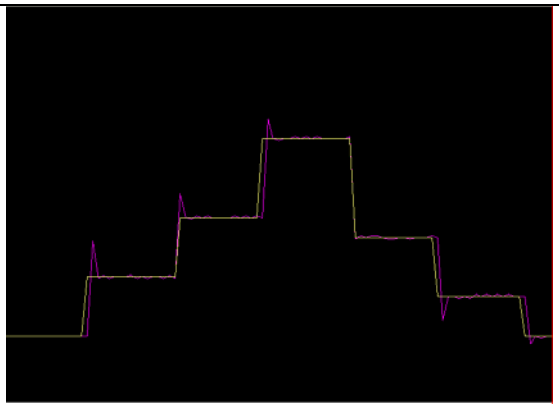
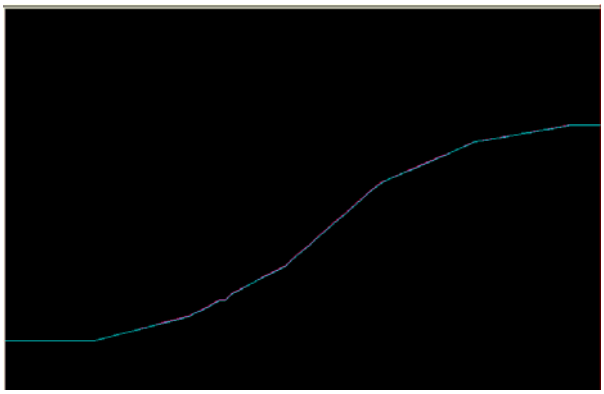
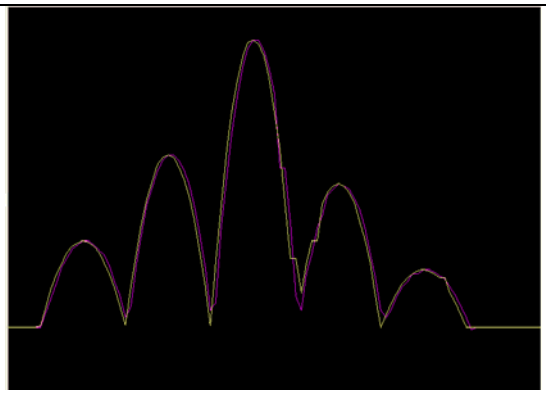
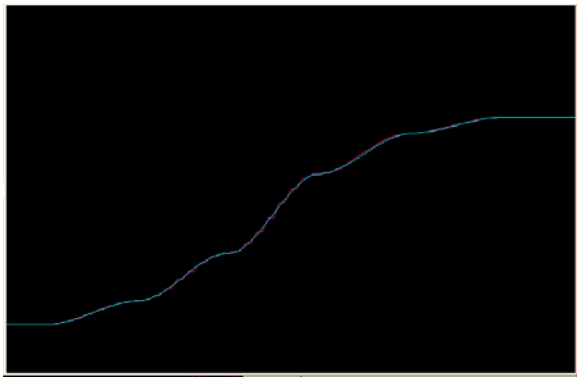
Punto	1	2	3	4	5
Posición [rot]	7.5	22.5	47.5	60	65
Tiempo [s]	0.5	1	1.5	2	2.5
Velocidad al final del punto	0	0	0	0	0

Tabla 3.13: Datos del primer perfil PVT

Punto	1	2	3	4	5
Posición [rot]	7.5	22.5	47.5	60	65
Tiempo [s]	0.5	1	1.5	2	2.5
Velocidad al final del punto	1000	2000	1800	900	0

Tabla 3.14 Datos del segundo perfil PVT

En la tabla 3.15 quedan resumidas las gráficas obtenidas en los ejemplos anteriores. Como se puede comprobar, los puntos de los tres ejemplos son los mismos, solo varían las velocidades. Posteriormente, al realizarse la interpolación, los perfiles resultantes son completamente distintos.

	Perfil de velocidad	Perfil de posición
PT		
PVT-1		

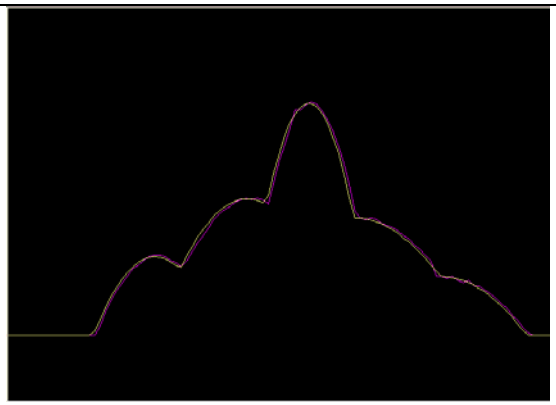
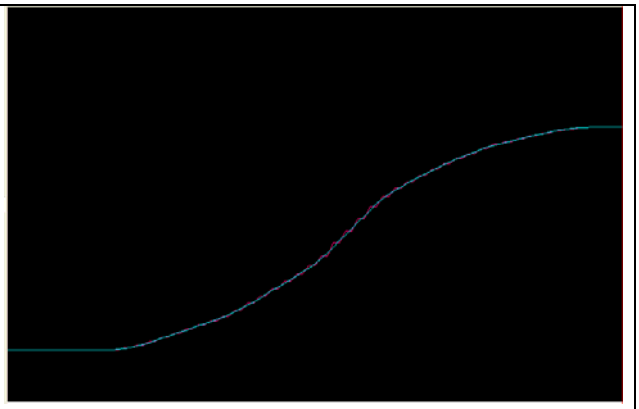
PVT-2		

Tabla 3.15: Perfiles generados por los modos PT y PVT

Como se puede observar en la tabla 3.15, el modo PT ejecuta un perfil de velocidad a base de escalones en el que las velocidades constantes que mantiene para llegar a la posición en el tiempo requerido se pueden calcular dividiendo la diferencia de rotaciones del punto anterior y el actual entre el tiempo pedido para llegar al punto. Viendo la gráfica del perfil de velocidad se puede observar que en este caso el motor oscila un poco debido a que tiene que ejecutar escalones. Esto no pasa de la misma forma en el modo PVT, donde se puede observar que dando un valor adecuado al parámetro de la velocidad final en cada punto (caso 2) se obtiene un perfil de velocidades en el que la variación de velocidad se realiza de forma progresiva. Por ello se consigue que las aceleraciones/deceleraciones no sean tan grandes al inicio de cada punto y se suavicen durante todo el perfil. Comparando los dos casos del modo PVT, se observa que dando unos valores incorrectos a la velocidad final se incurre en que el motor haga esfuerzos elevados creando picos de velocidad muy marcados.

Analizando los perfiles de posición, se puede ver que el generado por el modo PT consta de tramos rectos pertenecientes a la variación de velocidad requerida para cada punto, mientras, en el caso 2 del modo PVT se observa que el perfil resultante queda con variaciones de pendiente muy suaves. No ocurre lo mismo en el caso 1 del modo PVT donde se crean picos debidos a la errónea elección del parámetro velocidad final que fuerza al motor a alcanzar una velocidad absurda a la hora de crear el perfil.

Por otro lado, el modo PVT es más complicado de generar puesto que hay que dar el parámetro adicional de la velocidad final, el cuál tiene la dificultad añadida de que un valor erróneo puede crear un perfil que haga someter al motor a un sobreesfuerzo innecesario.

3.6 EL LENGUAJE TML

El Technosoft Motion Lenguage (TML) es el lenguaje que maneja el driver. Con él que se pueden implementar programas y funciones que se almacenan en la memoria EEPROM y posteriormente son llamadas desde el programa principal.

El TML es un lenguaje secuencial. Un programa completo almacenado en la memoria consta de dos partes, el setup y la parte de programación de los movimientos del motor.

Durante el setup, el driver adquiere la información del tipo de motor que tiene conectado y los parámetros de configuración de éste y los encoders o sensores de efecto Hall utilizados. Por otro lado, también procesa información relativa al número de eje que toma el motor, quedando definidas las protecciones contra cortocircuitos, sobreintensidades, etc. Todo esto se hace modificando los registros encargados de guardar la configuración o directamente se puede utilizar el IPM Motion Studio que genera el contenido de dichos registros usando un entorno más sencillo. Una vez que se inicia el driver, éste busca en la memoria un BEGUIN y lo ejecuta hasta que encuentra un ENDINIT. Con la ejecución del ENDINIT se hace una configuración básica del motor y de los sensores que no podrá ser cambiada hasta que haya un reset.

Después del setup se incorpora el programa principal, luego la tabla de vectores de interrupción, después las rutinas de servicio a la interrupción y, por último, las funciones.

El código de instrucciones de un programa en TML consiste de 1 a 5 palabras de 16 bits. La primera palabra es el código de operación que está dividido en dos partes: los bits del 15 al 9 indican el código de operación y el resto el ID del operando. Las palabras restantes llevan los datos.

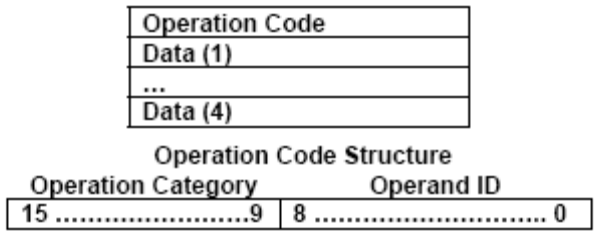


Figura 3.18: Formato de las instrucciones TML

El TML trabaja con distintos tipos de datos: registros de TML, variables TML, variables de usuario y parámetros de TML.

Las variables de usuario se las puede declarar con cualquier nombre y pueden ser de los siguientes tipos:

- int entero de 16 bits
- uint entero de 16 bits sin signo
- fixed "32 bits fixed-point data"
- long entero de 32 bits
- ulong entero de 32 bits sin signo

Para programar sólo se pueden utilizar las variables tipo int, fixed y long. Cada dato de TML tiene asociada una dirección de memoria, ésta puede ser asociada de forma directa o indirecta (usando un puntero).

Los registros pueden ser de 3 tipos:

- Registros de configuración: son los que guardan la configuración del motor y los sensores que se define durante el setup y también el modo de inicio del motor.
- Registros de comandos: estos registros activan/desactivan protecciones del software, configuran las interrupciones y las opciones de comunicación.
- Registros de estado: proporcionan información sobre las comunicaciones, el modo de funcionamiento del motor, las protecciones del sistema y las interrupciones del TML.

Los parámetros más utilizados durante la programación en TML son los siguientes:

- CPOS: (long) posición deseada (relativa o absoluta) en unidades de posición.
- CSPD: (fixed) velocidad deseada.
- CACC: (fixed) aceleración deseada.

Las variables usadas con mayor frecuencia son:

- TPOS: (long) posición objetivo
- TSPD: (fixed) velocidad objetivo
- TACC: (fixed) aceleración objetivo
- APOS: (long) posición actual
- ASPD: (long) velocidad actual

Otro elemento importante en la programación en TML son las instrucciones con los que se ejecutan acciones. Las instrucciones que más usadas en el lenguaje TML son:

- CPR: comando de posición relativa
- CPA: comando de posición absoluta
- MODE PPx: pone el modo de perfil de posición x (x= 0, 1, 2, 3)
- MODE SPx: pone el modo de perfil de velocidad x (x= 0,1)
- TUM1: genera la nueva trayectoria comenzando desde los valores actuales de posición y velocidad de referencia (p. ej. no actualizar los valores de referencia de la posición de motor y la velocidad)
- TUM0: genera la nueva trayectoria comenzando desde los valores actuales de posición y velocidad de referencia (p. ej. actualizar los valores de referencia de la posición de motor y la velocidad)
- UPD: actualizar el modo de movimiento y los parámetros. Iniciar el movimiento

Para más información sobre los parámetros, variables y registros ver el manual MotionChip II Configuration Setup User Manual donde se puede encontrar un listado con todos ellos, así como información relativa a su uso.

Con las variables, las instrucciones y los parámetros mencionados anteriormente ya se pueden crear pequeños programas de movimiento del motor. Otra forma más sencilla para crear programas en TML es utilizar la función Motion Wizard del IPM Motion Studio. A continuación se muestran un par de ejemplos creados a partir de la función Motion Wizard, uno en el que se realiza un control mediante la posición del motor y otro que se realiza mediante la velocidad:

- Ejemplo de control por posición:

```
CACC = 1.5; // command acceleration = 1.5
            // encoder counts/sampling2
CSPD = 20.; // command speed = 20 counts/sampling
CPOS = 20000; // command position = 20000 counts
CPA; // command position is absolute
MODE PP3; // set position profile mode 3
TUM1; // keep the position and speed reference
UPD; // update - start the motion
!MC; // set event on motion complete
```

WAIT!; // wait for the event to occur

- Ejemplo de control por velocidad:

CACC = 1; // command acceleration = 1.0 counts/sampling2

CSPD = -25.5; // command speed = -25.5 counts/sampling

// negative command speed = negative direction

MODE SP1; // set speed profile mode 1

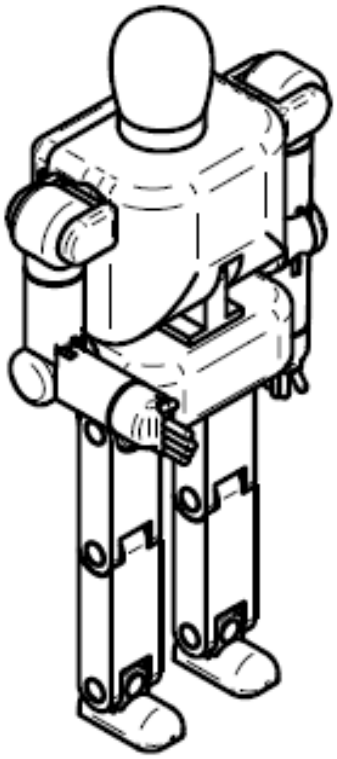
UPD; // update - start the motion

Hay otros muchos modos de funcionamiento pero no serán analizados en profundidad, puesto que el objetivo de este proyecto es establecer un control del motor usando el protocolo CANOpen quedando restringido el uso del lenguaje TML a llamadas de ejecución de funciones y programas. Por ello se va a proceder a la explicación de la creación de funciones.

Para crear una función, en la ventana de Proyect se va al apartado de Motion y dentro de él se hace click en Functions. Aparecerá una ventana donde se pueden añadir las funciones. Se escribe el nombre de la función y se pulsa en Add. Después de esto, la función añadida tiene que aparecer en el menú desplegable de Functions. Para editar la función se hace click en ella y se escriben los comandos que se quiere que incluya. Para llamar a dicha función desde el programa principal, se usa el comando CALL (nombre de la función).

Para obtener más información sobre programación en lenguaje TML, buscar en el manual correspondiente.

El programa EasyMotion Studio tiene una herramienta bastante interesante con la cuál se pueden traducir los comandos TML a distintos protocolos, como pueden ser TechnoCAN, TMLCAN, RS-232 y RS-485.



Capítulo 4: El CANbus

Introducción al bus de datos CAN. Descripción del protocolo CANopen y de los mensajes a enviar para controlar el driver.

4.1 EL BUS DE CAMPO CAN

CAN (Controller Area Network) es un bus de comunicación serie, originalmente desarrollado para la industria automovilística. El protocolo CAN abarca parte de los siete niveles del modelo de referencia ISO / OSI. La velocidad de transmisión de datos depende de la longitud del bus: 1Mbps para 40m, 125kbps para 500m y 10kbps para 5km. Provee dos tipos de servicio de comunicación: transmisión y petición de un mensaje. El resto de servicios que ofrece, como señalización de errores o el reenvío de tramas de datos erróneas son transparentes al usuario, es decir, el chip controlador de CAN lo realiza automáticamente.

CAN proporciona:

- Jerarquía multimaestro: permite construir sistemas inteligentes. Si un nodo de la red tiene un fallo, el resto de la red sigue operativa.
- Comunicación distribuida: el dispositivo que envía información lo hace a todos los nodos de la red, que la leen y deciden si es de interés para cada uno de ellos.
- Mecanismos sofisticados de detección de errores: permiten el reenvío automático en caso de error en la comunicación.

El hardware CAN cubre las dos primeras capas del modelo de referencia OSI, y las diferentes soluciones a nivel de software cubren los niveles siguientes. La siguiente figura nos muestra los niveles del modelo OSI:

Figura 4.1: Capas del modelo OSI

Como el hardware nos cubre solamente las dos primeras capas del modelo, surge la necesidad de buscar un sofware que nos cubra las restantes. En nuestro caso usaremos el protocolo CANOpen del que se hablará posteriormente.

El protocolo de CAN define la capa de enlace de datos y parte de la capa física del modelo OSI. La organización ISO ha definido un estándar que incorpora las especificaciones de CAN como parte de la capa física: codificación y decodificación de los bits, sincronización y temporización y señales físicas.

4.1.1 Codificación y decodificación de bits

El sistema elegido es NRZ (Non Return to Zero). El nivel de la señal permanece constante a lo largo del tiempo nominal de cada bit, y es el tiempo requerido para su representación. Otros métodos de codificación son por ejemplo Manchester y PWM.

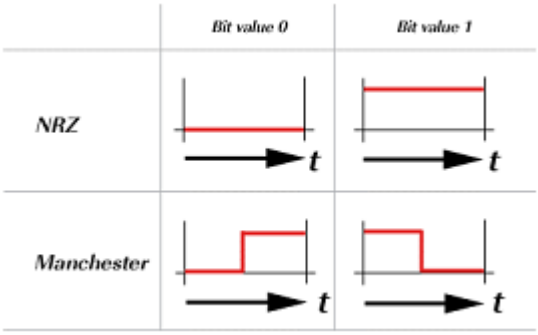


Figura 4.2: Modos de codificación de bits

Según este modo de codificación, la señal puede mantenerse constante a lo largo del tiempo (si hay una sucesión de bits de igual valor), por lo que la lectura debe realizarse asegurando que no se exceda el tiempo máximo correspondiente a cada bit. Esta consideración es importante para la sincronización. Si se produce un envío de cinco bits de igual valor se inserta en la trama un nuevo bit de relleno de nivel contrario, que debe ser eliminado por el receptor para que sea procesada de forma correcta la información enviada.

4.1.2 Sincronización y temporización

A nivel de bit, el bus CAN transmite de forma síncrona. Esto aumenta la capacidad de transmisión pero requiere un método sofisticado de sincronización. Mientras que la sincronización a nivel de bit en sistemas de transmisión basados en caracteres (asíncrona) se realiza mediante la recepción del bit de comienzo presente en cada carácter, en un protocolo de transmisión síncrona existe una serie de segmentos en cada tiempo nominal de bit. Para que el receptor haga una lectura correcta de la información, es necesario hacer una sincronización continua. Por tanto se incluyen segmentos de fase de memorización antes y después del punto de muestreo en el interior del intervalo de cada bit.

El protocolo de CAN regula el acceso al bus mediante un sistema de arbitraje inteligente. La propagación de la señal desde el transmisor al receptor y su vuelta al transmisor debe realizarse dentro del tiempo nominal de un bit. Por eso se añade al tiempo reservado por la sincronización un segmento de propagación de la señal, junto a los segmentos de fase de memorización.

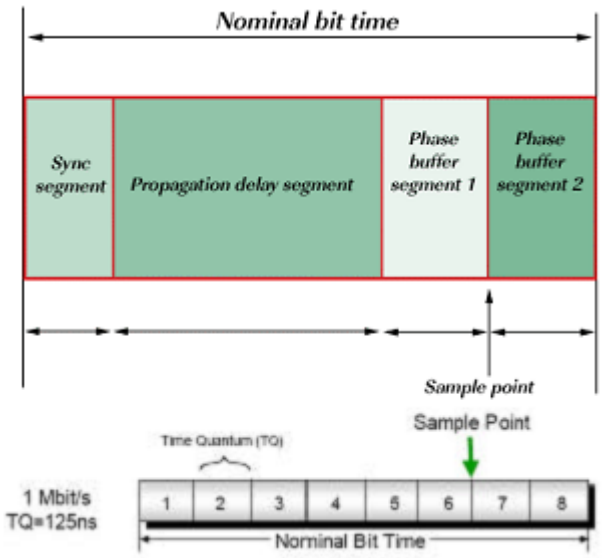


Figura 4.3: Esquema del tiempo nominal de cada bit

Se distinguen dos tipos de sincronización:

- Al inicio de la trama: el tiempo nominal de cada bit se reinicia al final del segmento de sincronización. Por eso el límite entre dos bits está ubicado en el interior del segmento de sincronización del nuevo bit.
- En el interior de la trama: se hace una resincronización que acorta o prolonga el tiempo nominal de cada bit fijando el punto de muestreo en función del límite detectado.

4.1.3 Dependencia tasa de transmisión – longitud del bus

En función del tamaño del segmento de propagación de la señal con una determinada tasa de transferencia se puede determinar la longitud máxima del bus, o la máxima tasa de transferencia dada la longitud del bus. El segmento de propagación de la señal viene determinado por la situación en el peor caso de comunicación, es decir, por los dos nodos que estén más alejados el uno del otro en el bus. Es el tiempo que emplea una señal en atravesar el bus de un nodo a otro y volver al nodo de inicio después de la sincronización. Sólo entonces el primer nodo puede decidir si es su propia señal la señal actual del bus o ha sido reemplazada por la señal de otro nodo. Esta consideración es importante para el arbitraje del bus.

La tabla siguiente recoge un resumen de las características temporales de una red CAN en función de su longitud y de la tasa de transmisión:

Bit Rate	Longitud del bus	Longitud "Time Quantum"	Tiempo Nominal de Bit	Localización Punto de muestreo	Latencia
1 Mbit/s	25 m	125 ns	8 t_q (1 µs)	6 t_q (750 ns)	130 µs
800 kbit/s	50 m	125 ns	10 t_q (1.25 µs)	8 t_q (1 µs)	162.5 µs
500 kbit/s	100 m	125 ns	16 t_q (2 µs)	14 t_q (1.75 µs)	260 µs
250 kbit/s	250 m	250 ns	16 t_q (4 µs)	14 t_q (3.5 µs)	520 µs
125 kbit/s	500 m	500 ns	16 t_q (8 µs)	14 t_q (7 µs)	1.04 ms
50 kbit/s	1000 m	1.25 µs	16 t_q (20 µs)	14 t_q (17.5 µs)	2.6 ms
20 kbit/s	2500 m	3.125 µs	16 t_q (50 µs)	14 t_q (43.75 µs)	6.5 ms
10 kbit/s	5000 m	6.25 µs	16 t_q (100 µs)	14 t_q (87.5 µs)	13 ms

Tabla 4.1: Temporización en el bus

4.1.4 Medios físicos, topología de red y acceso al bus

La base de la transmisión de mensajes en el bus CAN es la posibilidad de representar los valores de bit dominante y recesivo. Esto es posible tanto para medios eléctricos y ópticos. Para los medios ópticos el nivel recesivo se representa por "oscuridad" y el dominante como "luz". Para los medios eléctricos, la tensión de salida del bus y otros parámetros vienen definidos en ISO 11898-2, ISO 11898-3, SAE J2411 e ISO 11992.

El medio físico más comúnmente usado es un par trenzado con señales diferenciales y retorno común, aunque hay otros desarrollos como un único cable o la transmisión de señales por los cables de alimentación.

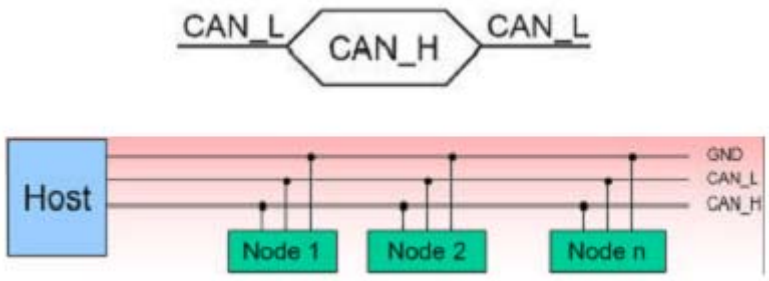


Figura 4.4: Arquitectura básica de una red CAN y señales eléctricas

Los parámetros de los medios eléctricos son muy importantes si la longitud del bus es considerable. La resistencia y la sección de los cables hacen que se vaya reduciendo los niveles de tensión a medida que aumenta la longitud del bus. La sección de los cables se calcula en función de la resistencia de entrada de los nodos conectados a la red, de la distancia entre los dos nodos más separados y del mínimo nivel de tensión que permite al receptor interpretar de forma correcta la información. Del mismo modo, asumiendo una longitud del bus determinada, existirá una tasa máxima de transferencia de datos en función del tiempo de propagación de la señal. Otro efecto poco deseable es el reflejo de la señal eléctrica al final del bus. Este efecto debe ser evitado para que cada nodo lea de forma correcta el nivel del bus en cada momento. Por eso se añaden en los extremos del bus unas resistencias terminales.

Cada dispositivo conectado al bus tiene un circuito integrado encargado del control y tratamiento de las señales transmitidas y recibidas. Con el objetivo de adecuar los niveles de tensión entre el bus y el controlador y de protección frente a cortocircuitos o niveles de tensión peligrosos en el bus, los controladores acceden al bus a través de un interfaz que consiste básicamente en un amplificador para la transmisión y otro para la recepción.

4.2 EL PROTOCOLO

El protocolo CAN es un estándar internacional definido en la norma ISO 11898. Además, para garantizar la compatibilidad de los circuitos controladores de CAN se ha definido la norma ISO 16845.

El protocolo CAN está basado en un mecanismo de comunicación distribuido, del tipo productor-consumidor, donde cualquier nodo de la red puede transmitir y recibir mensajes. Cada mensaje contiene un campo de arbitraje que incluye la dirección donde se envía y el tipo de mensaje. Todos los nodos de la red observan el estado del bus bit a bit y aceptan o rechazan el mensaje en función del identificador. Este esquema de comunicación permite un alto grado de modularidad, y hace posible la inserción de nuevos dispositivos en la red sin necesidad de hacer modificaciones hardware o software.

4.2.1 Transmisión de datos en tiempo real

En sistemas de tiempo real, hay datos que deben ser transmitidos con más urgencia y frecuencia que otros, como puede ser la velocidad de un motor frente a su temperatura. La prioridad con que un mensaje se transmite en comparación con otro menos urgente se especifica en el identificador de cada mensaje, expresado en sistema binario: el identificador con menor valor en binario tendrá mayor prioridad. El acceso al bus se resuelve mediante un sistema de arbitraje inteligente: el nivel dominante tiene prioridad sobre el recesivo. El nodo que intente escribir un bit recesivo pierde en este arbitraje frente a un nodo que intente escribir un bit dominante y se mantiene a la escucha, convirtiéndose en receptor del mensaje con mayor prioridad, y no pudiendo acceder al bus de nuevo mientras no esté disponible. En la figura 4.6 el nodo 1 pierde el arbitraje de acceso al bus frente a los nodos 2 y 3 al intentar escribir el bit 5 con nivel recesivo; y el nodo 3 lo pierde frente al 2 en el bit RTR.

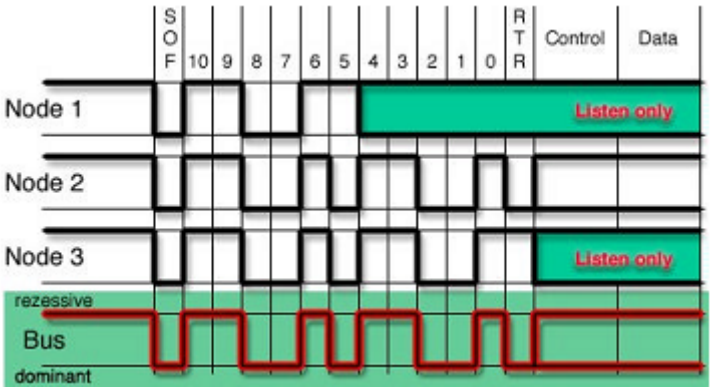


Figura 4.5: Resultado del arbitraje de acceso al bus

4.2.2 Formato del mensaje CAN

El protocolo CAN soporta dos formatos de mensaje, que básicamente se diferencian en la longitud del campo de arbitraje:

- **Formato base:** el campo de arbitraje contiene 11 bits. Es conocido formalmente como CAN 2.0 A. En este caso el bit IDE se envía como dominante.
- **Formato extendido:** el campo de arbitraje contiene 29 bits. Es conocido formalmente como CAN 2.0 B. En este caso el bit IDE se envía como recesivo.

En la misma red pueden coexistir mensajes de los dos formatos, pero en caso de colisión en el acceso al bus, tiene prioridad los mensajes de formato base. En cuanto a los controladores, existen de ambos tipos. Los que soportan formato extendido interpretan bien mensajes de formato base, pero al revés no ocurre igual.

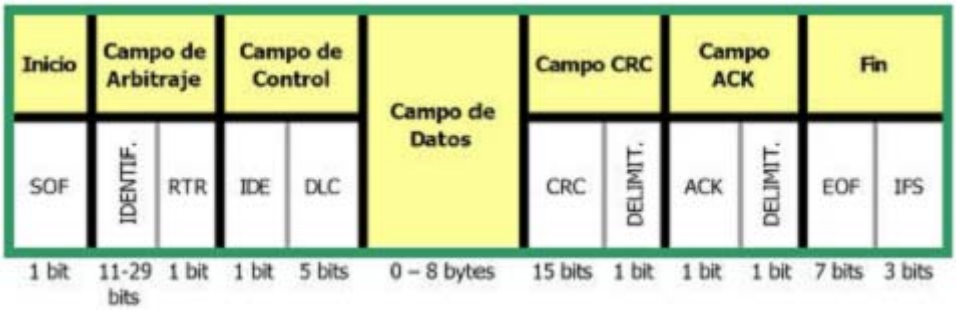


Figura 4.6: Estructura de un mensaje CAN

La transmisión de un mensaje CAN se hace de la siguiente forma:

- Envío del inicio del mensaje: SOF (Start Of Frame).
- Envío del Campo de Arbitraje: comprende el identificador del mensaje (11 bits para CAN 2.0 A, 29 bits para CAN 2.0 B) y el bit RTR (Remote Transmission Request, que se utiliza para distinguir entre un mensaje que contiene información y un mensaje que solicita información).
- Envío del Campo de Control: comprende el bit IDE (Identifier Extensión, para distinguir entre formato base y formato extendido) y el DLC (Data Lenght Code, que indica el número de bytes que contiene el siguiente campo).
- Envío del Campo de Datos: entre 0 y 8 bytes de datos.

- Envío del Campo CRC: Cyclic Redundant Check, código de redundancia que garantiza la integridad de la trama enviada. 15 bits seguidos de 1 bit delimitador.
- Envío del Campo ACK: Acknowledge. Comprende 2 bits: ACK Slot (1 bit), que se envía como recesivo y que es sobrescrito por el receptor con un bit dominante cuando recibe de forma correcta los datos, y 1 bit delimitador.
- Envío del fin del mensaje: se hace a través de los 7 bits del EOF (End Of Frame) y del IFS (Intermisión Frame Space), que es el mínimo número de bits que separa dos mensajes consecutivos. Es entonces cuando el bus queda libre para que otros nodos puedan acceder a él.

El protocolo CAN, de forma transparente al usuario, hace uso de diferentes mecanismos para la detección y señalización de errores:

- CRC (Cyclic Redundant Check).
- Verificación de la estructura del mensaje recibido.
- ACK (posibilidad de comunicar que se ha recibido el mensaje).

A nivel de bit, implementa dos mecanismos:

- Monitorización del bus por parte del nodo que envía el mensaje detectando las diferencias entre el bit enviado y el recibido.
- Bit de relleno: se añade a la trama un bit de nivel contrario por cada cinco bits iguales que se envían.

Si por alguno de estos mecanismos se detecta un error, se aborta la transmisión mediante el envío de un mensaje de error. De este modo se previene a los otros nodos para que no acepten información que pueda ser errónea, y automáticamente el nodo intenta transmitir de nuevo. En el caso de que un nodo envíe de forma continua mensajes de error, puede provocar el bloqueo de la red. El protocolo CAN dispone de un mecanismo que permite distinguir situaciones de error esporádicas de errores permanentes debido a un fallo en el nodo, y permite operar a la red de tal modo que los demás nodos no se vean afectados.

4.3. EL PROTOCOLO CANOPEN

CANOpen es un protocolo basado en la capa física del bus CAN, que cubre las demás capas del modelo OSI. Está orientado al control de movimiento de motores, aunque ahora es utilizada en otros muchos campos de la actividad industrial.

Cualquier dispositivo CANOpen puede entenderse como un dispositivo genérico, que está conectado a la red CAN por un lado y por otro lado a los datos de entradas y salidas específicas de la aplicación. La comunicación entre la red CAN y la aplicación se realiza mediante un Diccionario de Objetos. Este Diccionario de Objetos es único para cada dispositivo y representa un completo acceso de la aplicación implementada en cuanto a transmisión de datos y parámetros de configuración. El acceso al Diccionario de Objetos se hace mediante un protocolo de comunicación, consistente en distintas funciones para propósitos diferentes: objetos de comunicación estándares para transmisión en tiempo real (PDO, Process Data Objects), datos de configuración (SDO, Service Data Objects), funciones especiales (Time Stamp, Sync Message y Emergency Message) y datos de mantenimiento de la red (Boot-up Message, NMT Message y Error Control). Estos objetos conforman la interfaz de comunicación.

Un objeto del Diccionario de Objetos viene definido por un índice, para el que se disponen de 16 bits y un subíndice para el que se disponen de 8 bits, y es similar a una estructura de C:

Index	Subindex	Variable Accessed	Data Type
2100	0	NumberOfEntries	Unsigned8
2100	1	BaudRate	Unisgned16
2100	2	NumberOfDataBits	Unsigned8
2100	3	NumberOfStopBits	Unsigned8
2100	4	Parity	Unsigned8

C-Structure Equivalent:

```
typedef struct {
    UNSIGNED8     NumberOfEntries;
    UNSIGNED16    BaudRate;
    UNSIGNED8     NumberOfDataBits;
    UNISGNED8     NumberOfStopBits;
    UNSIGNED8     Parity;
} tRS232Param
```

Figura 4.7: Ejemplo de un objeto del Diccionario de Objetos

De este Diccionario de objetos, algunas entradas son comunes a todos los dispositivos y otras son específicas, tal y como muestra la siguiente tabla:

Índice	Objeto	
0000H	No se utiliza	COMÚN A TODOS LOS DISPOSITIVOS
0001H – 025FH	Tipos de Datos	
0260H – 0FFFH	Reservado	
1000H – 1FFFH	Área del Perfil de Comunicación	
2000H – 5FFFH	Área del Perfil Específico del Fabricante	ESPECÍFICO DEL DISPOSITIVO
6000H – 9FFFH	Área del Perfil Estandarizado del Dispositivo	
A000H – BFFFH	Área del Perfil Estandarizado del Interfaz	
C000H – FFFFH	Reservado	

Figura 4.8: Entradas del Diccionario de Objetos

4.4 IMPLEMENTACIÓN EN CANOPEN

En este capítulo se analizará más en profundidad las distintas opciones que nos ofrece el protocolo CANOpen para comunicarnos con nuestro driver y cuales son los campos que nosotros editaremos para generar mensajes útiles y reconocibles por el driver ya que su protocolo interno exige recibir mensajes con unos datos concretos para poder controlar los motores o para mandar mensajes de respuesta con la información solicitada.

Se utilizará el estándar 2.0 A de CANbus, con identificadores reducidos de 11 bits. El mensaje CAN completo constará de 130 bits máximos, dónde se incluyen los 64 bits máximos disponibles para datos. El esquema por defecto que tienen los mensajes es el siguiente:

- **COB-ID**: el COB-ID es el Identificador del Campo de Arbitraje de un mensaje CAN. Este campo recoge los 11 bits del identificador (por utilizarse el estándar 2.0A de CAN) y depende directamente del tipo de mensaje a enviar a través del bus de datos. Los 4 más significativos (del 8 al 11) corresponden al tipo de Objeto de Comunicación, y los 7 menos significativos corresponden a la dirección del dispositivo. Por lo tanto se pueden enviar 16 tipos de mensajes, y se pueden colocar en la red hasta 127 dispositivos. La tabla 4.2 recoge los distintos valores del COB-ID según el mensaje a enviar:

COB Type	Bits 8 - 11 of COB-ID	ID Range
NMT	0000	0
SYNC	0001	128 (80h)
Time Stamp	0010	256 (100h)
Emergency	0001	129…255 (81h…ffh)
PDO1 - Transmit	0011	385…511 (181h…1ffh)
PDO1 - Receive	0100	513…639 (201h…27fh)
PDO2 - Transmit	0101	641…767 (281h…2ffh)
PDO2 - Receive	0110	769…895 (301h…37fh)
PDO3 - Transmit	0111	897…1023 (381h…3ffh)
PDO3 - Receive	1000	1025…1151 (401h…47fh)
PDO4 - Transmit	1001	1153…1279 (481h…4ffh)
PDO4 - Receive	1010	1281…1407 (501h…57fh)
SDO - Transmit	1011	1409…1535 (581h…5ffh)
SDO - Receive	1100	1537…1663 (601h…67fh)
Error control (node guarding)	1110	1793…1919 (701h…77fh)

Tabla 4.2: Valores de COB-ID según el mensaje

- **Tamaño de datos**: indica el tamaño en bits que son datos útiles, es decir parámetros.

Tamaño parámetros	Valor
8 bits	2Fh
16 bits	2Bh
32 bits	23h
0 *	40h

Tabla 4.3: Bits del tamaño de datos

* los mensajes que no necesitan parámetros corresponden a mensajes de sólo lectura y aquellos que devuelven un mensaje con datos concretos. Un ejemplo son los objetos que informan de la velocidad o posición del motor.

- **Índice**: dirección del objeto con el que se quiere comunicar. Consta de 16 bits.

- **Subíndice:** en cada objeto pueden existir varias "mini funciones", identificadas por su correspondiente subíndice de 8 bits. En caso de no existir subíndices en el objeto, por defecto tomaría el valor 00h.

- **Parámetros**: son los datos que necesita el objeto como parámetros de entrada. Su tamaño varía en función del tipo de dato que maneje el objeto. Si un objeto no los necesitase o no usase los 32 bits máximos disponibles, por ejemplo objetos de sólo lectura, se completaría con todo '0'.

TAMAÑO PARÁMETROS	ÍNDICE	SUBÍNDICE	PARÁMETROS

Dependiendo del tipo de mensaje, este requerirá la utilización de unos campos u otros por lo que se han definido todos de forma general y más adelante se especificará cuales serán requeridos en cada tipo de mensaje.

Además, en la descripción del objeto aparece si el objeto es de sólo lectura, solo escritura, o de escritura y lectura; si se puede mapear en un PDO, el tamaño de los datos, etc.

Ejemplo de construcción de un mensaje

En el ejemplo enviamos un mensaje al objeto 60FFh (Target speed) para definir la velocidad del motor.

Object description:

Index	$60FF_h$
Name	Target velocity
Object code	VAR
Data type	INTEGER32

Figura 4.9: Descripción del objeto Target Velocity

El motor dispone de un encoder de 500 líneas incrementales, y siguiendo el manual, para establecer 500 rpm hay que introducir como parámetros 10AAACh. El índice del objeto es 60FFh y no tiene subíndice. Los datos de entrada o parámetros son de 32 bits, lo que corresponde al código 23h.

El mensaje en su formato de lectura MSB – LSB, sería:

TAMAÑO PARÁMETROS	ÍNDICE	SUBÍNDICE	PARÁMETROS
23	60 FF	00	10 AA AC

Tabla 4.4: Mensaje en formato MSB-LSB

Hay que tener en cuenta que el protocolo de CANbus sigue un método de lectura de mensajes contrario al habitual. Los datos se envían para que se lean desde el bit menos significativo al más significativo (LSB–MSB). Es decir, para construir el mensaje hay que intercambiar los bytes de posición dentro de cada campo.

TAMAÑO PARÁMETROS	ÍNDICE	SUBÍNDICE	PARÁMETROS
23	FF 60	00	AC AA 10

Tabla 4.5: Mensaje en formato CAN

Por lo tanto se enviaría un mensaje con COB-ID '6XX' (600 pues es el COB-ID del SDO y en XX se pone el id del nodo al que se manda el mensaje) y datos '0x23816000ACAA10'.

4.4.1 NMT

Un mensaje NMT es muy sencillo, pues sólo contiene dos bytes de datos y su COB-ID es 0. El primer byte de datos contiene el comando específico de gestión de la red y el segundo el identificador del nodo al que va dirigido. Si este segundo byte tiene valor 0, el mensaje va dirigido a todos los nodos de la red. Los comandos NMT se utilizan para controlar el estado de la comunicación del driver.

4.4.1.1 Estados del driver

El driver puede presentar los siguientes estados: Inicialización, Pre-Operacional, Operacional y Parado.

- **Inicialización**: después del encendido del dispositivo está en el estado Inicialización y automáticamente pasa al estado Pre-Operacional.
- **Pre-Operacional:** la secuencia de arranque ha sido completada pero no se ha recibido ningún comando para pasar al estado Operacional. El dispositivo sólo responderá a mensajes SDO y NMT, pero no a los PDO
- **Operacional:** el dispositivo está completamente disponible y puede responder a cualquier objeto de comunicación, tanto a mensajes SDO como a mensajes PDO.
- **Parado:** sólo puede responder a mensajes NMT.

El estado de Inicialización está dividido en tres sub-estados para permitir reiniciar el nodo de forma parcial o completa.

En el sub-estado Reiniciar Aplicación los parámetros del área específica del fabricante y del perfil estandarizado del dispositivo vuelven a su valor preestablecido, y en el sub-estado Reiniciar Comunicaciones lo hacen los parámetros del perfil de comunicaciones. El tercer sub-estado es Inicialización, y es donde los nodos de la red pasan propiamente al estado de Inicialización después de encenderse.

4.4.1.2 MENSAJES NMT

Los mensajes NMT pueden cambiar el estado de los nodos utilizando los siguientes servicios en los cuales el COB-ID es siempre 0, el primer byte siguiente indica el estado en el quiere que se sitúe y el segundo byte indica el nodo al que se dirige el mensaje. De los bytes que preceden al COB-ID, el primero toma un valor que viene predefinido según al estado al que se dirige el driver, y el segundo puede

tomar valores comprendidos entre 0 y 255 (00 y ff en hexadecimal). Los estados posibles son los siguientes (se usará para los ejemplos como destinatario el nodo 6):

- **Iniciar nodo remoto:** el cliente al mandar este mensaje pone al nodo seleccionado en modo operacional.

COB-ID	Estado	Nodo
0	01	06

Tabla 4.6: Mensaje para iniciar un nodo

- **Parar nodo remoto:** el cliente al mandar este mensaje pone al nodo seleccionado en modo parado. Para entrar en este modo, hay que enviar un comando de parada rápida o de detención con la palabra de control de la que se hablará posteriormente.

- **Entrar en modo pre-operacional:** el cliente al mandar este mensaje pone al nodo seleccionado en modo pre-operacional.

COB-ID	Estado	Nodo
0	80	06

Tabla 4.7: Mensaje para poner un nodo en modo pre-operacional

- **Resetear nodo:** el maestro pone al nodo esclavo en el sub-modo resetear aplicación. En este estado el driver realiza un reseteo de software y vuelve a entrar en modo pre-operacional.

COB-ID	Estado	Nodo
0	81	06

Tabla 4.8: Mensaje para resetear un nodo

- **Resetear comunicación:** se cargan de nuevo los parámetros de la comunicación y se pasa al estado pre-operacional.

COB-ID	Estado	Nodo
0	82	06

Tabla 4.9: Mensaje para resetear un nodo

4.4.2 Mensajes de sincronización

Otro tipo de mensajes, los de sincronización, son parecidos a los NMT pero tienen funciones para la sincronización. Además, el COB-ID de estos mensajes deja de ser 0.

El mensaje de sincronización permite la sincronización de los dispositivos en la red y consigue que la transmisión de PDOs sea síncrona. La periodicidad de estos mensajes se define mediante una herramienta de configuración durante el proceso de reinicio. Estos mensajes tienen un identificador perteneciente al grupo de muy alta prioridad, el 128, y no lleva ningún dato. Está disponible en el índice 1005h del Diccionario de Objetos.

Durante la ejecución de aplicaciones críticas, que requieren una sincronización más exacta, los drivers pueden usar el protocolo de sincronización opcional de alta resolución, que emplea un formato especial de mensaje. El índice de objeto 1013h contiene sellos de tiempo con resolución de 1µs. Este objeto puede ser mapeado en un PDO para definir un mensaje alta resolución. El PDO debería ser configurado para la transmisión síncrona. Cuando uno de los drivers es puesto como maestro de sincronización, el sello de tiempo de Alta resolución por defecto es enviado usando el COB-ID definido en el objeto COB-IDs de los sellos de tiempo de alta resolución de índice 2004h.

4.4.3 Mensajes de emergencia

Un driver envía un mensaje de la emergencia (EMCY) cuando ocurre un error interno. El mensaje de emergencia es transmitido sólo una vez por cada error. Mientras no ocurra ningún nuevo error, el driver no mandará más mensajes de emergencia. Puede que ningún dispositivo sea consumidor de este mensaje, pero pueden ser varios los receptores. El COB-ID con el que se envían los mensajes de emergencia se define con el objeto 1014h. La acción a tomar por el receptor de un mensaje de emergencia es específica de cada aplicación. CANOpen define varios códigos de error de emergencia para ser transmitidos en un mensaje de emergencia, que tiene sólo ocho bytes de datos con la siguiente información:

Código del error de emergencia	Registro de errores (objeto 1001h)	Campo de error específico del fabricante
0-1	2	3-7

Tabla 4.10: Formato del mensaje de emergencia

Los detalles en cuanto a las condiciones que pueden generar mensajes de emergencia son presentados en el objeto de índice 2000h "registro de error de movimiento".

Para comprobar los errores existentes se puede acceder al objeto 1001h correspondiente al registro de errores. Comprende 1 byte de datos y según los bits que estén activos denotarán un error u otro:

Bit	Descripción
0	Error genérico
1	Corriente
2	Voltaje
3	Temperatura
4	Error de comunicación
5	Perfil de dispositivo específico
6	Reservado (siempre a 0)
7	Específico del fabricante

Tabla 4.11: Descripción de los bits del registro de errores

4.4.4 SDO

Los SDO (Service Data Objects) son usados por el maestro de CANOpen para tener acceso a cualquier objeto del Diccionario de Objetos del driver. Los SDOS son usados normalmente para configurar el driver después de ser encendido, para mapear PDOs y para comunicación poco frecuente de baja prioridad. El protocolo de transmisión SDO permite transmitir objetos de cualquier tamaño, por lo que puede ser necesario el envío y recepción de varios segmentos, siendo necesaria la confirmación de recepción. El primer byte del primer segmento contiene la información del control del flujo de datos, los siguientes tres bytes el índice y el subíndice del objeto dentro del diccionario al que se desea acceder y los otro cuatro bytes están disponibles para datos del usuario, todo ello precedido del COB-ID pertinente formado por 11 bits. El segundo y siguientes segmentos enviados, utilizando el mismo COB-ID, contienen un byte de control y siete bytes de datos del usuario. El receptor de los mensajes confirma cada segmento o bloque de segmentos recibido, por lo que es un sistema de comunicación punto a punto.

El formato que tienen los mensajes SDO es el siguiente:

COB-ID	CONTROL FLUJO DE DATOS (1 byte)	ÍNDICE (2 bytes)	SUBÍNDICE (1 byte)	DATOS (4 bytes)

Tabla 4.12: Formato de los mensajes SDO

- Los 4 bits más significativos del COB-ID (del 8 al 11) corresponden al tipo de Objeto de Comunicación, y los 7 menos significativos corresponden al nodo del driver.
- Con el objeto 1200h subíndice 01h se accede al COB-ID del SDO transmisor cuyo valor es 600, y en el subíndice 02h del mismo objeto se accede al COB-ID del SDO receptor en cuál tiene de valor 580.

Por ello, el COB-ID de los paquetes de SDO recibidos por el driver se calcula como 600h + el ID del nodo. El COB-ID de los paquetes de SDO enviados por el driver es calculado como 580h + el ID del nodo. El byte de control de flujo de datos toma los valores indicados en el apartado 4.4 de este proyecto.

4.4.4.1 MENSAJES SDO

Imaginando que se quisiera enviar un mensaje de consulta (valor 40h en el byte de control de flujo de datos) al nodo 6 para acceder al objeto 6041h de subíndice 00h, quedaría el siguiente mensaje:

COB-ID	Control de flujo de datos	Índice de objeto	Subíndice	Datos
606 (mensaje SDO de transmisión: 600h + 6h)	40 (mensaje de consulta)	41 60	00	00 00 00 00

Tabla 4.13: Ejemplo de un mensaje de consulta

Si por el contrario se quisiese enviar 32 bits de datos (por ejemplo 0010AAACh) al objeto 6040h y subíndice 01h al nodo 6, el mensaje quedaría así:

COB-ID	Control de flujo de datos	Índice de objeto	Subíndice	Datos
606 (mensaje SDO de transmisión: 600h + 6h)	23 (32 bits de datos)	40 60	01	AC AA 10 00 (0010AAACh transformado a formato CAN)

Tabla 4.14: Ejemplo de un mensaje de 32 bits de datos

4.4.4.2 SDOS para el control y estados del driver

La máquina de estados describe el estado del driver y las secuencias de control posibles. Los estados del driver pueden ser cambiados con la palabra de control (control word) y/o de acuerdo con eventos internos. El estado actual del driver se refleja en la palabra de estado (Status Word).

La siguiente figura representa los estados posibles en los que se puede encontrar el driver:

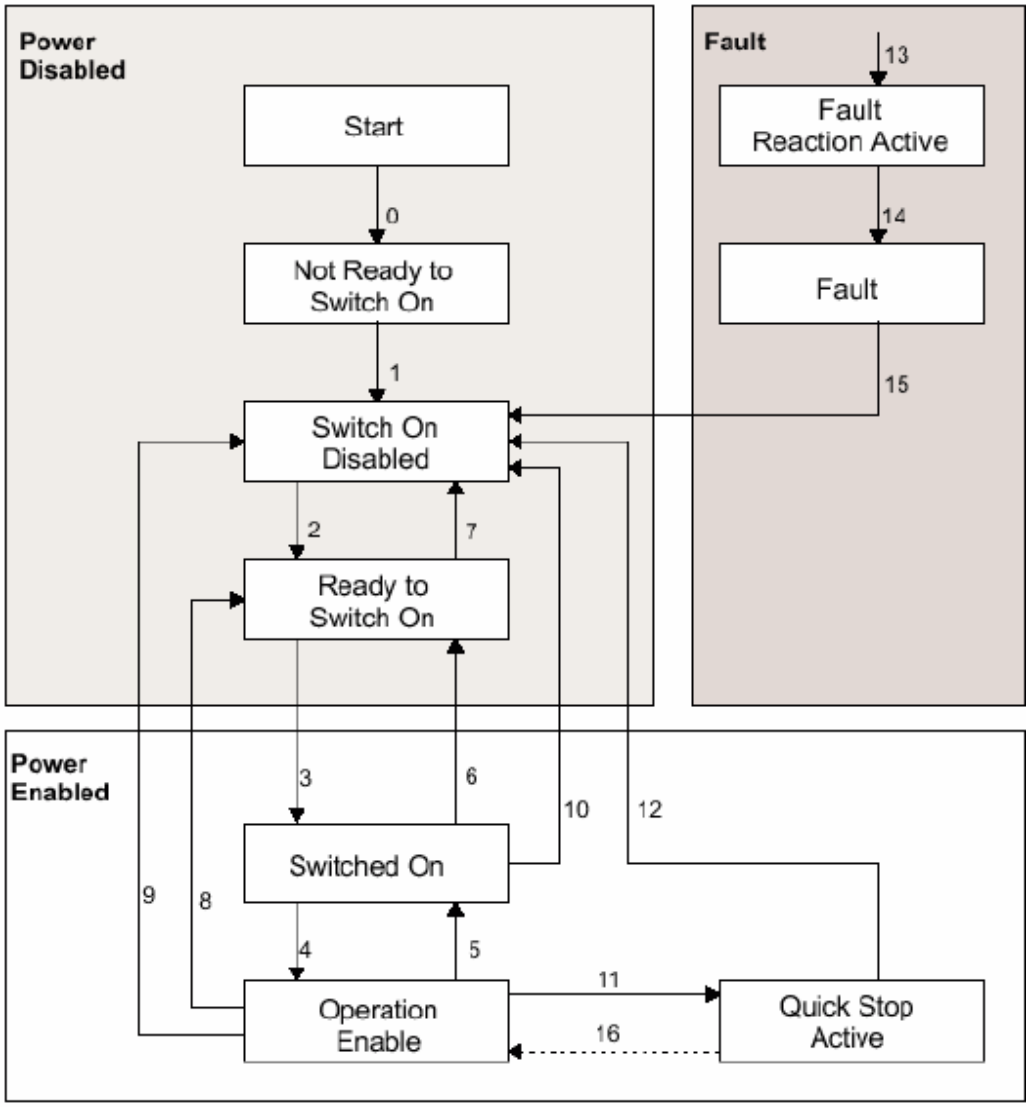


Figura 4.10: Máquina de estados del driver

- **Palabra de Control.**

La Palabra de Control es el objeto de índice 6040h. Con ella se puede controlar la máquina de estados del driver. Esto se utiliza para habilitar/deshabilitar su alimentación, comenzar/parar los movimientos y para sacarlo del estado de fallo. La Palabra de Control consta de 16 bits y la siguiente tabla muestra el significado de cada uno de ellos:

Bit	Valor	Descripción
15	0	Modo de registro inactivo
	1	Modo de registro activo
14	0	Cuando se realiza una actualización, se actualizan los valores de velocidad y posición
	1	Cuando se realiza una actualización, no se actualizan los valores de velocidad y posición
13		Cuando está a 1 cancela la ejecución de la función de TML llamada por el objeto 2006h. El bit automáticamente es reinicializado por el driver cuando se ejecuta la orden.
12	0	Sin acción
	1	Si el bit 14 = 1 pone la posición demandada a 0 Si el bit 14 = 0 pone la posición actual a 0
11		El significado de este bit depende del modo de funcionamiento del driver.
10-9		Reservado
8	0	Sin acción
	1	El motor frena
7	0	Sin acción
	1	Reseteo de los fallos. En la transición de 1 a 0 de este bit se resetean los fallos.
4-6		Modo de operación
3		Habilitar operación
2		Parada rápida
1		Habilitación del voltaje
0		Encendido

Tabla 4.15: Descripción de los bits de la Palabra de Control

Ejemplo de acceso a la Palabra de Control

Se accede vía SDO al objeto 6040h en el que se quiere almacenar los siguientes datos: 00 00 5F 70h. El mensaje resultante sería el siguiente:

COB-ID	Control de flujo de datos	Índice de objeto	Subíndice	Datos
606 (mensaje SDO de transmisión: 600h + 6h)	2B (16 bits de datos)	40 60	00	70 5F 00 00 (los 16 bits de datos y los 16 restantes se rellenan con bits '0')

Tabla 4.16: Ejemplo de acceso a la Palabra de Control

- **Palabra de Estado**

Como se mencionó anteriormente, con el objeto 6041h se accede a la Palabra de Estado. Con este objeto se puede consultar el estado actual en que se encuentra el driver. Su tamaño es de 2 bytes. Para acceder a este objeto hay que mandar un mensaje al índice 6041h. La tabla 4.17 muestra como queda el mensaje a enviar. La tabla muestra 4.18 el significado de cada uno de los bits que conforman la Palabra de Estado.

COB-ID	Control de flujo de datos	Índice de objeto	Subíndice	Datos
606 (mensaje SDO de transmisión: 600h + 6h)	40 (mensaje de consulta)	41 60	00	00 00 00 00 (todos los bits a '0')

Tabla 4.17: Ejemplo de acceso a la Palabra de Estado

Bit	Valor	Descripción
15	0	Eje apagado. El control del motor no se realiza
	1	Eje encendido. El control del motor se realiza
14	0	Ningún evento activo o el evento programado no ha ocurrido aún
	1	Evento ocurrido
13-12		Modo de Operación Específico
11		Límite interno activo
10		Objetivo alcanzado
9	0	El driver está en el modo local y no ejecutará la orden
	1	Los parámetros del driver pueden ser modificados vía CAN y el driver puede ejecutar la orden.
8	0	Ninguna función de TML o de homing ha sido ejecutada. La ejecución de la última función de TML llamada o de homing fue completada.
	1	Se está ejecutando una función de TML o de homing. Hasta que no termine no se puede ejecutar otra.
7	0	Sin peligro
	1	Una función de TML o de homing fue llamada mientras se estaba ejecutando una, la última llamada se ha ignorado.
6		Encendido deshabilitado
5		Parada rápida. Cuando este bit está a cero se ejecuta una parada rápida
4	0	El voltaje de suministro de motor está presente
	1	El voltaje de suministro de motor no está presente
3		Si está a 1 hay o hubo un error en el driver
2		Operación permitida
1		Encendido
0		Preparado para encenderse

Tabla 4.18: Descripción de los bits de la Palabra de Estado

4.4.5 PDO

Los PDO (Process Data Objects) se usan para envíos de mensajes de alta prioridad, en tiempo real entre el maestro CANOpen y los drivers. Los PDOs envían mensajes sin confirmación puesto que la recepción del mensaje queda garantizada por el protocolo de CAN. Durante una transmisión de datos, el productor envía un TPDO (Transmit PDO) con un identificador que corresponde con el identificador del mensaje RPDO (Receive PDO) de uno o más consumidores. El driver ISCM8005 acepta 4 TPDOs y 4 RPDOs. El contenido de los PDOS puede ser editado según las necesidades de aplicación gracias al mapeo de PDOs dinámico. Esta operación se realiza durante la fase de configuración del driver usando SDOs.

UN PDO queda definido por dos tipos de objetos: el Objeto de Comunicación y el Objeto de Mapeo. El Objeto de Comunicación define el COB-ID del PDO, el tipo de transmisión y el acontecimiento que provoca la transmisión. El Objeto de Mapeo contiene las descripciones de los objetos mapeados en el PDO, por ejemplo el índice, el subíndice y el tamaño.

La siguiente tabla muestra los objetos que hacen referencia a los 4 TPDOs y los 4 RPDOs, indicando también cuál es el objeto correspondiente a la comunicación y cual al mapeo:

	TPDO1	**TPDO2**	**TPDO3**	**TPDO4**	**RPDO1**	**RPDO2**	**RPDO3**	**RPDO4**
Comunicación	1800h	1801h	1802h	1803h	1400h	1401h	1402h	1403h
Mapeo	1A00h	1A01h	1A02h	1A03h	1600h	1601h	1602h	1603h

Tabla 4.19: Índices de los RPDO Y TPDO

Los Objetos de Comunicación de los RPDO contienen en el subíndice 00h el número de entradas que tiene el objeto, en el subíndice 01h el COB-ID del PDO y en el subíndice 02h se define el carácter de recepción del PDO. Esto mismo ocurre con los Objetos de Comunicación de los TPDO, a diferencia que tienen un subíndice adicional, el 05h con el que se ajusta un timer para eventos.

Los Objetos de Mapeo de los RPDO y TPDO contienen en el subíndice 00h el número de entradas válidas dentro del registro de mapeo. Este número de entradas es también el número de los objetos que serán transmitido/recibidos en el correspondiente PDO. Los subíndices de 01h al número de entradas contienen la información sobre los objetos mapeados. En estos subíndices se describe el contenido del PDO por su índice, subíndice y longitud. Para cambiar el mapeo de un PDO, primero el PDO tiene que ser deshabilitado (el objeto 160Xh con subíndice 00h se pone a 0). Después de eso, los objetos pueden ser mapeados de nuevo.

4.4.5.1 Procedimiento para mapear un PDO

1. Se deshabilita el PDO. En el Objeto de Mapeo de PDO (índices 1600h-1603h para RPDOs y 1A00h- 1A03h para TPDOs) se pone el primer subíndice a 0.
2. Si fuera necesario, se cambian los parámetros de comunicación del PDO (índices 1400h-1403h para RPDOs y 1800h-1803h para TPDOs): el COB-ID, el tipo de transmisión o el acontecimiento que provoca la transmisión.
3. Se mapean los nuevos objetos. Se escribe en el Objeto de Mapeo de PDO'S (de índices 1600h-1603h para RPDOs y 1A00h-1A03h para TPDOs) en los subíndices 1 a 8 la descripción de los objetos que serán mapeados. Se pueden mapear hasta 8 objetos de 1 byte de tamaño.
4. Se habilita el PDO. En el subíndice 0 del Objeto de Mapeo asociado al PDO (índices 1600h-1603h para RPDOs y 1A00h-1A03h para TPDOs) se escriben el número de objetos mapeados.

Ejemplo de mapeo de un PDO

Se quiere mapear el PDO3 receptor de un nodo con ID 6 con Palabra de Control (índice 6040h) y Modo de Operación (índice 6060h).

1) Deshabilitado del RPDO: Se escribe 0 en el objeto 1602h, subíndice 0

COB-ID	Control de flujo de datos	Índice de objeto	Subíndice	Datos
606h (mensaje SDO + ID del nodo)	2F (8 bits de datos)	02 16	00	00 00 00 00

2) Supongamos que los parámetros de comunicación están bien.

3) Se mapean los nuevos objetos:

   a) Se escribe en el objeto 1602h, subíndice 01h la descripción de Palabra de Control:

COB-ID	Control de flujo de datos	Índice de objeto	Subíndice	Datos
606	23	02 16	01	10 00 40 60

b) Se escribe en el objeto de índice 1602h, subíndice 02h la descripción de Modo de Operación:

COB-ID	Control de flujo de datos	Índice de objeto	Subíndice	Datos
606	23	02 16	02	08 00 60 60

4) Se habilita el PDO: Se escribe 2 en el objeto 1602h, subíndice 00h

COB-ID	Control de flujo de datos	Índice de objeto	Subíndice	Datos
606	2F	02 16	00	02 00 00 00

4.4.5.2 PDOS para el control y estados del driver

Otra forma de controlar los estados del driver es mandando un mensaje PDO que acceda directamente a la máquina de estados. Estos PDOs dejan de seguir el esquema que venían teniendo la mayoría de los PDO y SDO para asemejarse más a los NMT. Estos mensajes constan de los 11 bits del COB-ID (en el que se incluye la dirección del nodo) y 2 bytes de datos en los que se define el estado al que se quiere cambiar el driver. Unos ejemplos de los posibles mensajes que se pueden enviar son:

1) Para cambiar el estado del driver a preparado para encenderse

COB-ID	206 (COB-ID del PDO + ID del nodo)
Datos	06 00

2) Para cambiar el estado del driver a encendido

COB-ID	Datos
206	07 00

3) Para cambiar el estado a operación permitida

COB-ID	Datos
206	0F 00

4.4.6 Modos de control

Este apartado pretende ahondar en los modos de control que posee el driver y en la programación de cada uno de ellos. Primeramente hay que seleccionar el modo en que se quiere que esté el driver. Para ello está el objeto Modos de Control (índice 6060h). Es un objeto de solo escritura por lo que no se puede leer de él. Según el valor que se escriba, el driver entrará en un modo de control u otro:

Valor	Descripción
-128…-6	Reservado
-5	Modo de Momento de rotación de Referencia Externo
-4	Modo de Velocidad de Referencia Externa
-3	Modo de Posición de Referencia Externa
-2	Modo de posición ECAM (Electronic Camming)
-1	Modo de posición EG (Electronic Gearing)
0	Reservado
1	Modo de perfil de posición
2	Reservado
3	Modo de perfil de velocidad
4, 5	Reservado
6	Modo homing
7	Modo de posición interpolado
8…127	Reservado

Tabla 4.20: Valores del objeto Modos de Control

Ejemplo de cambio de modo de control

Se quiere poner el driver en modo de control por velocidad, para ello accede vía SDO al objeto 6060h en el que se graba el valor 3 en el primer byte de datos. El mensaje resultante sería el siguiente:

COB-ID	Control de flujo de datos	Índice de objeto	Subíndice	Datos
606	2F	60 60	00	03 00 00 00

Tabla 4.21: Mensaje para cambiar de modo de control

Nota: para ver el estado actual del driver se utiliza el objeto 6061h mandando un mensaje de consulta

4.4.6.1 Modo de perfil de posición

Como ya se dijo en otro apartado de este capítulo, dentro de este modo de funcionamiento hay 2 tipos de perfiles:

- Perfil trapezoidal
- Perfil en "S"

Ambos modos se controlan con los mismos objetos:

- Posición objetivo (índice 607Ah). La posición objetivo es la posición a la cual el driver debería moverse usando los ajustes actuales de los parámetros de control de movimiento como la velocidad, la aceleración, y el tipo de perfil de movimiento. Viene dado en unidades de posición.
- Perfil de velocidad (índice 6081h). En el perfil de posición, este objeto representa la velocidad máxima que alcanza al final de la rampa de aceleración.
- Perfil de aceleración (índice 6083h). Representa los ratios de aceleración y deceleración usados para cambiar la velocidad entre dos niveles.

El modo de perfil en curva utiliza también el objeto "jerk time" de índice 2023h.

En el modo de control de posición, la palabra de control y la palabra de estado tienen los siguientes significados:

- Palabra de control:

Ver 6040h	Modo de Operación	Ver 6040h	Detención	Ver 6040h	Abs/Rel	Cambio inmediato	Nuevo set-point	Ver 6040h
15-12	11	10-9	8	7	6	5	4	3-0

Tabla 4.22: Estructura de la Palabra de Control en el modo de posición

El significado de los bits es:

Nombre	Valor	Descripción
Modo de operación	0	Perfil trapezoidal: si está en modo relativo no añade el nuevo punto a la posición demandada Perfil en "S": Detiene el movimiento con un perfil en "S"
	1	Perfil trapezoidal: si está en modo relativo añade el nuevo punto a la posición demandada Perfil en "S": Detiene el movimiento con un perfil trapezoidal
Nuevo set-point	0	No actualiza los parámetros de posición
	1	Actualiza los parámetros de posición
Cambio inmediato	0	Termina el movimiento actual y después comienza el siguiente
	1	Interrumpe el movimiento actual y comienza el siguiente.
Abs/Rel	0	La posición actual es un valor absoluto
	1	La posición actual es un valor relativo
Detención	0	Ejecuta el movimiento
	1	Para el driver

Tabla 4.23: Significado de los bits de la Palabra de Control en el modo de posición

- Palabra de estado:

Ver 6041h	Error de seguimiento	Reconocimiento de set-point	Ver 6041h	Objetivo alcanzado	Ver 6041h
15-14	13	12	11	10	9-0

Tabla 4.24: Estructura de la Palabra de Estado en el modo de posición

El significado de los bits es el siguiente:

Nombre	Valor	Descripción
Objetivo alcanzado	0	Detención=0 posición objetivo no alcanzada Detención=1 driver decelera
	1	Detención=0 posición objetivo alcanzada Detención=1 velocidad del driver es 0
Reconocimiento de set-point	0	El generador de trayectoria aceptará el nuevo set-point
	1	El generador de trayectoria no aceptará el nuevo set-point
Error de seguimiento	0	Sin error de seguimiento
	1	Error de seguimiento

Tabla 4.25: Significado de los bits de la Palabra de Estado en el modo de posición

Ejemplo de control en el modo de posición

1) Iniciar el nodo 6 mandando un mensaje NMT

COB-ID	Datos
0	01 06

2) Se cambia el estado del nodo de encendido deshabilitado a preparado para encenderse (ver objeto 6041h) mediante un PDO

COB-ID	Datos
206	06 00

3) Se cambia el estado del driver a encendido

COB-ID	Datos
206	07 00

4) Se cambia el estado a operación permitida vía PDO

COB-ID	Datos
206	0F 00

5) Modo de operación: se selecciona modo de posición (objeto 6060h, octavo bit a 1)

COB-ID	Control de flujo de datos	Índice de objeto	Subíndice	Datos
606	2F	60 60	00	01 00 00 00

6) Posición objetivo: en el objeto 607Ah, se ponen las rotaciones que se quieren expresadas en cuentas del encoder, por ejemplo 4 rotaciones corresponde a 00001F40h

COB-ID	Control de flujo de datos	Índice de objeto	Subíndice	Datos
606	23	7A 60	00	40 1F 00 00

7) Velocidad objetivo: en el objeto 6081h, se pone la velocidad deseada que alcanzará al final de la rampa de aceleración, por ejemplo 0010AAACh.

COB-ID	Control de flujo de datos	Índice de objeto	Subíndice	Datos
606	23	81 60	00	AC AA 10 00

8) Iniciar el perfil en modo absoluto.

COB-ID	Datos
206	1F 00

9) Esperar a que el movimiento acabe

4.4.6.2 MODO DE POSICIÓN INTERPOLADA

En el Modo de Posición Interpolada, se hace una interpolación de la posición con el tiempo para tener un control mas preciso cuando se manejan uno o varios drivers simultáneamente y se requiere una mayor precisión o coordinación entre ellos.

Modos internos. La siguiente figura muestra los estados existentes dentro del Modo de Posición Interpolada:

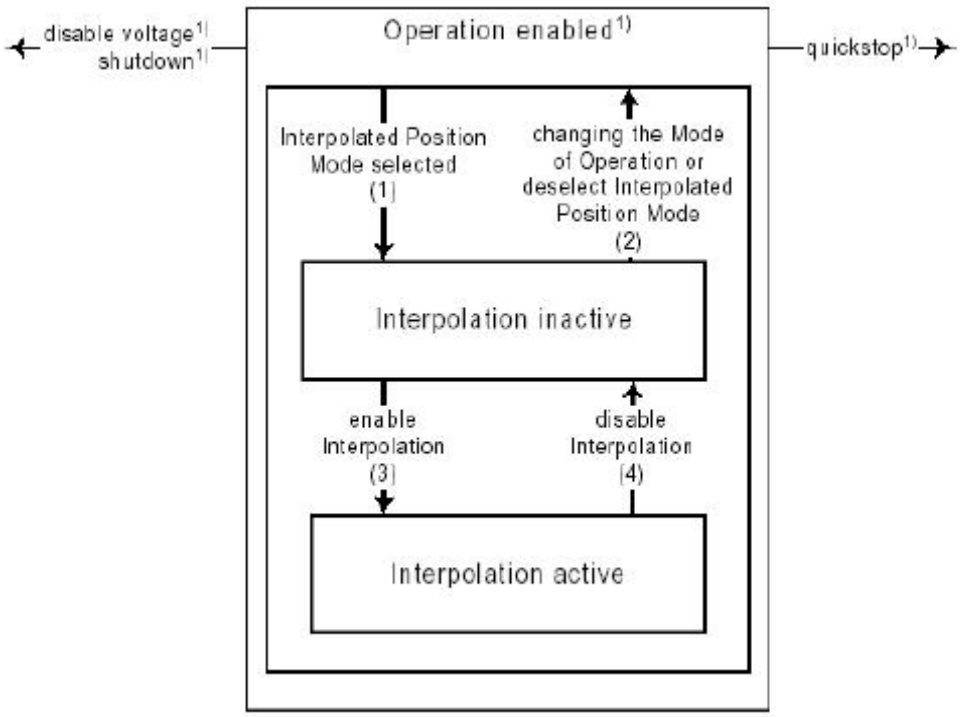


Figura 4.12: Modos internos del Modo de Posición Interpolada

En el Modo de Posición interpolada, la Palabra de Control y la Palabra de Estado adquieren el siguiente significado:

- Palabra de Control:

Ver 6040h	Opción de parada	Ver 6040h	Detención	Ver 6040h	Abs/Rel	Reservado	Habilitar el modo ip	Ver 6040h
15-12	11	10-9	8	7	6	5	4	3-0

Tabla 4.26: Estructura de la Palabra de Control en el Modo de Posición Interpolada

Significado de los bits:

Nombre	Valor	Descripción
Habilitar modo ip	0	Modo interpolación inactivo
	1	Modo interpolación activo
Abs/Rel	0	La posición actual es un valor absoluto
	1	La posición actual es un valor relativo
Detención	0	Ejecuta el movimiento
	1	Para el driver
Opción de parada	0	En la transición al modo inactivo, el driver se para inmediatamente utilizando el perfil de aceleración.
	1	En la transición al modo inactivo, el driver se para después de ejecutar el segmento actual.

Tabla 4.27: Significado de los bits de la Palabra de Control en el M. de Pos. Interpolada

- Palabra de estado:

Ver 6041h	Reservado	Modo ip activo	Ver 6041h	Objetivo alcanzado	Ver 6041h
15-14	13	12	11	10	9-0

Tabla 4.28: Estructura de la Palabra de Estado en el Modo de Posición Interpolada

Significado de los bits:

Nombre	Valor	Descripción
Objetivo alcanzado	0	Detención=0: posición objetivo no alcanzada Detención=1 driver decelera
	1	Detención=0 posición objetivo alcanzada Detención=1 velocidad del driver es 0
Modo ip activo	0	Modo interpolación inactivo
	1	Modo interpolación activo

Tabla 4.29: Significado de los bits de la Palabra de Estado en el M. de Pos. Interpolada

Para manejar el modo de posición interpolada, se utilizan los siguientes objetos:

- **Selección del Submodo de Interpolación** (índice 60C0h) En el Modo de Posición Interpolada, hay dos modos de funcionamiento: PT (Posición - Tiempo) que genera una interpolación lineal entre los puntos y el modo PVT (Posición-Velocidad-Tiempo) que hace una interpolación cúbica entre puntos. El submodo de funcionamiento solo puede ser cambiado mientras el driver está en el modo de interpolación inactivo. El objeto tiene de tamaño 16 bits, si se graba como dato -1, el driver entra en modo PVT y si se graba el valor 0 el driver entrará en modo PT

- **Registro de Datos de Interpolación** (índice 60C1h). Este objeto contiene las palabras de datos que son necesarias para realizar el algoritmo de interpolación. En el subíndice 01h se encuentra el primer parámetro de la función de interpolación (X1) y en el subíndice 02h se encuentra el segundo (X2). Para utilizar estos parámetros hay que mapearlos en alguno de los RPDO disponibles.

Con los parámetros X1 y X2 se forma una estructura de 64 bits. La forma de ésta depende del submodo en el que se encuentre el driver:

a) En el modo PVT se almacenan 4 parámetros:

   - Posición: 24 bits que toman valores de tipo long integer que representa la posición objetivo.
   - Velocidad: 24 bits que toman valores de tipo fixed que representa la velocidad que toma cuando llega a la posición objetivo.

- Tiempo: 9 bits que toman valores de tipo unsigned integer que representa el tiempo del segmento PVT.
- Contador: 7 bits que toman valores de tipo unsigned integer. Se puede utilizar para saber cuál fue el último punto que fue mandado al driver.

Byte 0	Posición [7…0]	
Byte 1	Posición [15…8]	
Byte 2	Velocidad [7…0]	
Byte 3	Posición [23…16]	
Byte 4	Velocidad [15…8]	
Byte 5	Velocidad [23…16]	
Byte 6	Tiempo [7…0]	
Byte 7	Contador [6…0]	Tiempo [8]

Tabla 4.30: Estructura de los bits en un mensaje PVT

b) En el modo PT se almacenan 3 parámetros:

- Posición: 32 bits que toman valores de tipo long integer que representa la posición objetivo.
- Tiempo: 16 bits que toman valores de tipo unsigned integer que representa el tiempo del segmento PVT.
- Contador: 7 bits que toman valores de tipo unsigned integer. Se puede utilizar para saber cuál fue el último punto que fue mandado al driver.

Byte 0	Posición [7…0]	
Byte 1	Posición [15…8]	
Byte 2	Posición [23…16]	
Byte 3	Posición [31…24]	
Byte 4	Velocidad [15…8]	
Byte 5	Tiempo [7…0]	
Byte 6	Tiempo [15…8]	
Byte 7	Contador [6…0]	Reservado

Tabla 4.31: Estructura de los bits en un mensaje PT

- **Estado del Modo de Posición Interpolada** (índice 2072h). Con este objeto se tiene acceso al estado en que se encuentra el modo de posición. Es un objeto de sólo lectura que transmite 2 bytes de datos con el siguiente significado:

Bit	Valor	Descripción
15	0	El buffer no está vacío
	1	El buffer está vacío
14	0	El buffer no está en un nivel bajo
	1	El buffer está en un nivel bajo (el número de putnos es menor o igual que el limite inferior establecido en el objeto 2074h)
13	0	El buffer no está lleno
	1	El buffer está lleno
12	0	No hay error en el contador
	1	Error en el contador
11	0	Sólo para modo PVT-El driver ha mantenido el modo de interpolación después de una condicion de buffer vacío (la velocidad del último punto era 0)
	1	Sólo para modo PVT-El driver ha ejecutado una parada rápida en el modo de interpolación después de una condicion de buffer vacío ya que la velocidad del último punto no era 0
10…7		Reservado
6….0		Valor actual del contador

Tabla 4.32: Significado de los bits del objeto Estado del Modo de Pos. Interpolada

- **Longitud del Buffer** en el Modo de Posición Interpolada (índice 2073h). A través de este objeto se puede cambiar el valor de la longitud por defecto del buffer. Cuando se escribe en este objeto, el buffer automáticamente resetea su contenido y se reinicia con la nueva longitud.

- **Configuración del Buffer** en el Modo de Posición Interpolada (índice 2074h). Con este objeto se puede controlar más en detalle la configuración del buffer:

Bit	Valor	Descripción
15	0	Nada
	1	Limpia el buffer y reinicializa las variables internas
14	0	Activa el comprobador del contador
	1	Desactiva el comprobador del contador
13	0	No cambia nada del contador
	1	Cambia el contador a los valores especificados en los bits de 0 a 6
12	0	Si el driver está en modo absoluto (bit 6 de la palabra de control está 0), la posición inicial es leída del objeto 2079h. Esto se usa para calcular la distancia para llegar al primer punto de PVT.
	1	Si el driver está en modo absoluto (bit 6 de la palabra de control está 0), la posición inicial es el valor actual de la posición demandada. Esto se usa para calcular la distancia para llegar al primer punto de PVT.
11…8		Nuevo parámetro para la señalización baja del buffer. Cuando el número de entradas en el buffer es igual o menos que el valor bajo del buffer, el bit 14 del objeto 2072h se pondrá a 1.
7	0	Sin cambios en los parámetros bajos del buffer
	1	El parámetro bajo del buffer toma el valor especificado en los bits 11…8
6…0		Nuevo valor del contador

Tabla 4.33: Significado de los bits del objeto Configuración del Buffer

- **Posición Inicial** en el Modo de Posición Interpolada (índice 2079h). A través de este objeto se le puede asignar una posición inicial al driver si el está en posicionamiento absoluto.

Ejemplo de control en el modo de posición interpolada

En este ejemplo se va a ejecutar de un movimiento en modo PVT que consta de 4 segmentos.

1) Iniciar el nodo 6 mandando un mensaje NMT

COB-ID	Datos
0	01 06

2) Se cambia el estado del nodo de encendido deshabilitado a preparado para encenderse (ver objeto 6041h)

COB-ID	Datos
206	06 00

3) Se cambia el estado del driver a encendido

COB-ID	Datos
206	07 00

4) Se cambia el estado a operación permitida vía PDO

COB-ID	Datos
206	0F 00

5) Se desactiva el RPDO2: se escribe '0' en el objeto 1601h, subíndice 00h

COB-ID	Control de flujo de datos	Índice de objeto	Subíndice	Datos
606	2F	01 16	00	00 00 00 00

6) Se mapean lo nuevos objetos:

   a) En el objeto 1601h, subíndice 1, se graba la descripción del subíndice 1 del objeto Registro de datos de interpolación (60C1h).

COB-ID	Control de flujo de datos	Índice de objeto	Subíndice	Datos
606	23	01 16	01	20 01 C1 60

b) En el objeto 1601h, subíndice 2, se graba la descripción del subíndice 2 del objeto Registro de datos de interpolación.

COB-ID	Control de flujo de datos	Índice de objeto	Subíndice	Datos
606	23	01 16	02	20 02 C1 60

7) Se habilita el RPDO2: se escribe 2 en el objeto 1601h, subíndice 00h

COB-ID	Control de flujo de datos	Índice de objeto	Subíndice	Datos
606	2F	01 16	00	02 00 00 00

8) Se selecciona el Modo de Posición Interpolada (en el objeto 6060h, valor 07h en el primer byte de datos)

COB-ID	Control de flujo de datos	Índice de objeto	Subíndice	Datos
606	2F	60 60	00	07 00 00 00

9) Selección del submodo PVT (se accede al objeto 60C0h y se le da el valor FFFFh en 16-bit.

COB-ID	Control de flujo de datos	Índice de objeto	Subíndice	Datos
606	2B	C0 60	00	FF FF 00 00

10) Se configura la longitud del buffer (objeto 2074h)

COB-ID	Control de flujo de datos	Índice de objeto	Subíndice	Datos
606	2B	74 20	00	00 0C 00 00

11) Se edita la posición inicial: se ponen 0.5 rotaciones en cuentas del encoder (500 líneas) que corresponden al valor 3E8h en el objeto 2079h

COB-ID	Control de flujo de datos	Índice de objeto	Subíndice	Datos
606	23	79 20	00	E8 03 00 00

12) Se envía el primer punto que queremos que tenga estas características:

Posición: 002710h

Velocidad: 000355h

Tiempo: 1F4h (nota: el '1' representa únicamente un 1 en el octavo bit)

Contador: 00h (nota: en este caso sólo son 7 bits a cero)

Los datos corresponden con el siguiente esquema:

Byte 0	Posición [7…0]		10
Byte 1	Posición [15…8]		27
Byte 2	Velocidad [7…0]		55
Byte 3	Posición [23…16]		00
Byte 4	Velocidad [15…8]		03
Byte 5	Velocidad [23…16]		00
Byte 6	Tiempo [7…0]		F4
Byte 7	Contador [6…0]	Tiempo [8]	01

Tabla 4.34: Esquema de los datos en el mensaje PVT

Finalmente el mensaje a mandar es el siguiente (suponiendo que se manda un PDO al nodo 6):

COB-ID	Datos
306	10 27 55 00 03 00 F4 01

13) Se envía el segundo punto. Hay que tener en cuenta el incremento del contador.

COB-ID	Datos
306	10 27 55 00 03 00 F4 03

14) Se envía el tercer punto

COB-ID	Datos
306	10 27 55 00 03 00 F4 05

15) Se envía el último punto

COB-ID	Datos
306	E0 B1 00 FF 00 00 F4 07

16) Se inicia un movimiento relativo

COB-ID	Datos
206	5F 00

Después de ejecutar esta serie de comandos, el motor debería realizar el siguiente movimiento:

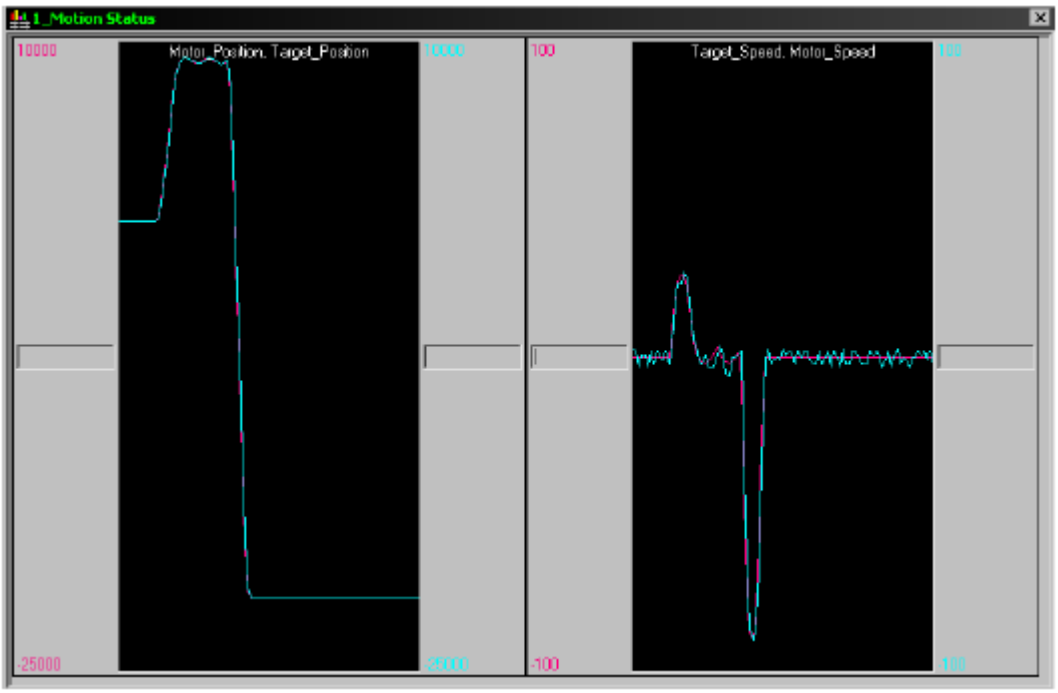


Figura 4.12: Perfil ejecutado por el motor

4.4.6.3 MODO DE PERFIL DE VELOCIDAD

En el Modo de Perfil de Velocidad el driver realiza un control sobre la velocidad. El generador de referencia calcula el perfil de velocidad con una forma trapezoidal, debido a una aceleración limitada. Este modo queda definido por el objeto de Velocidad Objetivo (índice 60FFh) que especifica la velocidad que ha de alcanzar el driver (el signo de velocidad especifica la dirección) y por el objeto de Perfil de Aceleración (índice 6083h) que indica el ratio de aceleración/desaceleración. Mientras este modo esté activo, cualquier cambio en objeto 6083h realizado por el maestro de CANOPEN actualizará la velocidad de demanda del driver, pudiéndose cambiar en marcha la velocidad de giro y/o la tarifa de aceleración/desaceleración.

El movimiento seguirá hasta que el bit de detención de la Palabra de Control sea puesto a 1. Un modo alternativo de parar el movimiento es poner la velocidad a cero. Mientras el modo de velocidad esté activo, siempre que se escriba dentro del objeto Velocidad Objetivo, el objeto Velocidad Demandada del driver se actualizará.

Para seleccionar el Modo Perfil de Velocidad se accede al objeto Modos de Control del Motor (índice 6060h) y se escribe el valor 3.

En el Modo de Perfil de Velocidad, la Palabra de Control y la Palabra de Estado adquieren el siguiente significado:

- Palabra de Control:

Ver 6040h	Detención	Ver 6040h	Reservado	Ver 6040h
15-9	8	7	6-4	3-0

Tabla 4.35: Estructura de la Palabra de Control en el Modo de Velocidad

Significado de los bits:

Nombre	**Valor**	**Descripción**
Detención	0	Ejecuta el movimiento
	1	Para el driver

Tabla 4.36: Significado de los bits de la Palabra de Control en el Modo de Velocidad

- Palabra de estado:

Ver 6041h	Máximo error de disminución	Velocidad	Ver 6041h	Objetivo alcanzado	Ver 6041h
15-14	13	12	11	10	9-0

Tabla 4.37: Estructura de la Palabra de Estado en el Modo de Velocidad

Significado de los bits:

Nombre	Valor	Descripción
Objetivo alcanzado	0	Detención=0: velocidad objetivo no alcanzada Detención=1 driver decelera
	1	Detención=0 velocidad objetivo alcanzada Detención=1 velocidad del driver es 0
Velocidad	0	La velocidad no vale 0
	1	La velocidad vale 0
Máximo error de disminución	0	Máxima disminución no alcanzada
	1	Máxima disminución alcanzada

Tabla 4.38: Significado de los bits de la Palabra de Estado en el Modo de Velocidad

Ejemplo de control en el modo de velocidad

1) Iniciar el nodo 6 mandando un mensaje NMT

COB-ID	Datos
0	01 06

2) Se cambia el estado del nodo de encendido deshabilitado a preparado para encenderse (ver objeto 6041h) mediante un PDO

COB-ID	Datos
206	06 00

3) Se cambia el estado del driver a encendido

COB-ID	Datos
206	07 00

4) Se cambia el estado a operación permitida mediante un PDO

COB-ID	Datos
206	0F 00

5) Modo de operación: se selecciona modo de velocidad (objeto 6060h con valor 03h)

COB-ID	Control de flujo de datos	Índice de objeto	Subíndice	Datos
606	2F	60 60	00	00 03 00 00

6) Velocidad objetivo: en el objeto 60FFh, se ponen la rotaciones por minuto que se quieren expresadas en cuentas del encoder, por ejemplo 600 rpm corresponden a 00140000h.

COB-ID	Control de flujo de datos	Índice de objeto	Subíndice	Datos
606	23	FF 60	00	00 00 14 00

7) Se ejecuta.

COB-ID	Datos
206	5F 00

4.4.7 Intercambio de datos maestro-driver

Una de ellas es el intercambio de datos entre el maestro y el driver, como pueden ser el valor de variables a las que no se puede acceder directamente mediante un objeto de CANOpen. Si se mira en el diccionario de objetos, se puede observar que para conocer la posición o la velocidad actual hay un objeto a través del cuál se puede hacer una consulta y conocer su valor. Sin embargo, para conocer el valor del flujo de intensidad que hay entre el driver y el motor no hay forma alguna de averiguarla.

Con los objetos 2064h, 2065h y 2066h, no sólo se puede obtener el valor de variables almacenadas en registros, sino que también se puede modificar el valor en algunos de ellos.

- **Lectura/Escritura del Registro de Configuración** (índice 2064h). Este objeto se usa para controlar la lectura de la memoria del driver y para escribir en el driver funciones de memoria. Este objeto contiene la dirección de memoria que se usará en la operación de lectura/escritura. También se puede especificar con este objeto el tipo de memoria que se va a usar (EEPROM, datos o programa) y el tipo de datos que se va usar en la siguiente lectura/escritura. Además, se puede especificar si se debería realizar un incremento de la dirección de memoria después de la operación de lectura o escritura. El autoincremento de la dirección de memoria es importante para ahorrar tiempo valioso en caso de que se quiera descargar de programa al driver así como cuando se quiera leer un gran bloque de datos a través del controlador. La siguiente tabla nos muestra la descripción del significado de cada bit del objeto:

Bit	Valor	Descripción
31…16	X	Dirección de memoria de 16 bits que se va a usar en la siguiente operación.
15..8	0	Reservado (valor siempre 0)
7	0	Auto-incremento de la dirección de memoria después de la lectura/escritura
	1	No Auto-incremento
6…4	0	Reservado (valor siempre 0)
3,2	00	El tipo de memoria seleccionada es la memora del programa
	01	El tipo de memoria seleccionada es la memora de datos
	10	El tipo de memoria seleccionada es la memora EEPROM
	11	Reservado
1	0	Reservado (valor siempre 0)
0	0	El tamaño de la siguiente operación de lectura/escritura es 16 bits
	1	El tamaño de la siguiente operación de lectura/escritura es 32 bits

Tabla 4.39: Descripción de los bits del objeto Lectura/Escritura del Reg. de Conf.

- **Escritura de 16/32 bits de datos** (índice 2065h). La función de este objeto es la de escribir valores de 16 o 32 bits con los parámetros definidos en el objeto 2064h (Lectura/Escritura en el Registro de Configuración). Después del correcto proceso de escritura, la dirección de memoria especificada en el objeto 2064h (bits 31…16) serán auto-incrementados o no, según como esté definido en el mismo registro. La siguiente tabla nos muestra la descripción del significado de los bits de este objeto:

Bit	Valor	Descripción
31…16	0	Reservado si el valor del bit 0 del objeto 2064h vale 0 (operación con variables de 16 bits)
	X	Los 16 MSB (more significant bits) de los datos si el bit 0 del objeto 2064h vale 1 (operación con variables de 32 bits)
15…0	X	Los 16 LSB (less significant bits) de los datos

Tabla 4.40: Descripción de los bits del objeto Escritura de 16/32 bits de datos

- **Lectura de 16/32 bits de datos** (índice 2066h). La función de este objeto es la de leer valores de 16 o 32 bits con los parámetros definidos en el objeto 2064h (Lectura/Escritura en el Registro de Configuración). Después del correcto operación de lectura, la dirección de memoria especificada en el objeto 2064h (bits 31…16) serán auto-incrementados o no, según como esté definido en el mismo registro. La siguiente tabla nos muestra la descripción del significado de los bits de este objeto:

Bit	Valor	Descripción
31…16	0	Reservado si el valor del bit 0 del objeto 2064h vale 0 (operación con variables de 16 bits)
	X	Los 16 MSB (more significant bits) de los datos si el bit 0 del objeto 2064h vale 1 (operación con variables de 32 bits)
15…0	X	Los 16 LSB (less significant bits) de los datos

Tabla 4.41: Descripción de los bits del objeto Lectura de 16/32 bits de datos

Ejemplo de lectura de registros del driver

Se va a mostrar un ejemplo en el que se lee el valor de un registro. Como en el caso del humanoide puede ser importante conocer el valor de la intensidad que está consumiendo el motor, se va a ejecutar un perfil de velocidad y se va a enviar un mensaje de consulta para obtener el valor de la corriente. El procedimiento es como sigue:

1) Iniciar el nodo 6 mandando un mensaje NMT

COB-ID	Datos
0	01 06

2) Se cambia el estado del nodo de encendido deshabilitado a preparado para encenderse (ver objeto 6041h) mediante un PDO

COB-ID	Datos
206	06 00

3) Se cambia el estado del driver a encendido

COB-ID	Datos
206	07 00

4) Se cambia el estado a operación permitida mediante un PDO

COB-ID	Datos
206	0F 00

5) Ahora hay que configurar el objeto Lectura/Escritura del Registro de Configuración (índice 2064h) para que cuando se ejecute la lectura, nos de la información de la intensidad. Para ello hay que buscar en el manual MotionChip II Configuration Setup User Manual, en el apartado de variables TML, la dirección de memoria de nuestra variable que en este caso es:

4.3.4.6. IQ. Q axis current

ADDRESS: 0x0230

Figura 4.13: Dirección de memoria de la variable IQ

Los datos del objeto se rellenarán de la siguiente forma para que el objeto se refiera a la dirección de memoria 0x0230h, no realice auto-incremento del valor de la dirección de memoria a la que se refiere, use la memoria de datos y el tamaño de los datos que contiene la variable sea de 32 bits:

Datos	0230h	00h	1	000	01	0	1
Bits	31-16	15-8	7	6-4	3-2	1	0

El mensaje a enviar queda de la siguiente forma:

COB-ID	Control de flujo de datos	Índice de objeto	Subíndice	Datos
606	23	64 20	00	85 00 30 02

Tabla 4.42: Mensaje para configurar el registro de lectura de variables

6) Modo de operación: se selecciona modo de velocidad (objeto 6060h con valor 03h)

COB-ID	Control de flujo de datos	Índice de objeto	Subíndice	Datos
606	2F	60 60	00	03 00 00 00

7) Velocidad objetivo: en el objeto 60FFh, se ponen la rotaciones por minuto que se quieren, expresadas en cuentas del encoder, por ejemplo 600 rpm corresponden a 00140000h.

COB-ID	Control de flujo de datos	Índice de objeto	Subíndice	Datos
606	23	7A 60	00	00 00 14 00

8) Se ejecuta

9) Se manda un mensaje de consulta al objeto Lectura de 16/32 bits de datos (índice 2066h)

COB-ID	Control de flujo de datos	Índice de objeto	Subíndice	Datos
606	40	66 20	00	00 00 00 00

Tabla 4.43: Mensaje de lectura de variables

Nota: Ver en anexo 2 resultados obtenidos en pruebas de consulta de intensidad

4.4.8 Opciones de control avanzadas

En este apartado se van a mostrar dos funcionalidades muy útiles que tiene el driver. Una de ellas es poder hacer llamadas desde el maestro CANOpen, a cualquiera de las 10 funciones TML almacenadas en la memoria del driver. Se pueden almacenar más de 10 funciones, pero solo es posible llamar a dicha cantidad. La segunda va un paso más allá y permite llamar a un programa entero de código TML almacenado en la memoria EEPROM del driver y ejecutarlo.

4.4.8.1 Llamadas a funciones TML

Resulta interesante poder almacenar funciones en la memoria EEPROM del driver que posteriormente puedan ser llamadas por el maestro CANOpen. Con esto se consigue descentralizar aún más el control del humanoide y así poder cerrar el lazo de control más fácilmente.

- **Llamar Función TML** (índice 2006h). El objeto permite la ejecución de una función de TML previamente descargada. Cuando se escribe en este objeto, se llama a la función de TML con el índice especificado en el valor proporcionado. El cuerpo de la función de TML se define usando EasyMotion Studio y guardándola en la memoria EEPROM del driver. El índice de la función sirve para ser ordenada en una tabla. No es posible llamar a otra función de TML, mientras la anterior todavía se está ejecutando. Se pueden llamar hasta 10 funciones a través de este mecanismo.

Ejemplo de una llamada a una función TML

Antes de realizar una llamada, el driver ha de tener almacenada en su memoria EEPROM la función que quiere ser llamada. Para ello, una vez creada en el programa EasyMotion se ha de ir al menú Aplication -> Motion -> Download program para descargarlo a la memoria. Una vez hecho esto, basta con mandar el siguiente mensaje desde el maestro CANOpen para llamar a una de las 10 funciones:

COB-ID	Control de flujo de datos	Índice de objeto	Subíndice	Datos
606	2B	06 20	00	01 00 00 00 (en el primer byte se pone el valor de 1 a C para llamar las funciones 1 a 10)

Tabla 4.44: Mensaje para llamar a una función TML

4.4.7.3 Llamadas a programas TML

El concepto de control distribuido puede dar un paso más hacia delante pudiéndose preparar y descargar en el driver un programa completo TML incluyendo funciones, procedimientos de homing, etc. La ejecución del programa TML puede ser iniciada simplemente escribiendo un valor en el objeto dedicado.

- **Ejecutar Programa TML** (índice 2077h): Este objeto se utiliza para ejecutar un programa TML almacenado en la memoria RAM o EEPROM del driver. El programa TML se puede descargar usando el software de EasyMotion Studio o bien mediante el maestro CANOpen usando el archivo .sw creado por el EasyMotion Studio. La escritura de cualquier valor en este objeto (con protocolo SDO) provocará la ejecución del programa TML en el driver.

Ejemplo de una ejecución de un programa completo de TML

El procedimiento para llamar a un programa es bastante similar al de una llamada a una función TML. Antes de realizar una llamada, el driver ha de tener almacenado en su memoria EEPROM el programa a ser llamado. Para ello, una vez creado en el programa EasyMotion se ha de ir al menú Aplication -> Motion -> Download program para descargarlo a la memoria. Una vez hecho esto, basta con mandar el siguiente mensaje desde el maestro CANOpen para llamar al programa:

COB-ID	Control de flujo de datos	Índice de objeto	Subíndice	Datos
606	2B	77 20	00	00 00 00 00

Tabla 4.45: Mensaje para ejecutar un programa TML

Nota: para poder realizar las llamadas a las funciones y el programa hay que mandar previamente dos comandos TML, AXISON y ENDINIT usando los objetos del apartado 4.4.7 para poner los bits 13 y 0 del registro MSR a 1.

4.4.9 Otros mensajes

- **Posición Actual del Motor** (índice 6064h).Con este objeto se comprueba el valor de la posición actual del motor.

Ejemplo de utilización del objeto

COB-ID	Control de flujo de datos	Índice de objeto	Subíndice	Datos
606	40	64 60	00	00 00 00 00

Tabla 4.46: Mensaje para obtener la posición actual

- **Posición Demandada** (índice 6062h). Comprueba el valor de la posición demandada.

Ejemplo de utilización del objeto

COB-ID	Control de flujo de datos	Índice de objeto	Subíndice	Datos
606	40	62 60	00	00 00 00 00

Tabla 4.47: Mensaje para obtener la posición demandada

4.5 TIEMPOS DE TRANSMISIÓN

En las comunicaciones entre el maestro y los drivers es necesario tener un control de los tiempos que se tarda en mandar un mensaje como por ejemplo cargar un punto de una trayectoria y recibir una respuesta con la lectura de la posición. Este es un apartado crítico ya que los tiempos de transmisión/recepción van a definir el tipo de tramas que se envía a través del bus según si da tiempo o no a cerrar los ciclos. Además, se parte con el inconveniente de que la velocidad máxima del bus CAN es de sólo 1Mbps que, en comparación con las velocidades máximas de otros buses como el SERCOS, es bastante reducida.

Las pruebas de comunicación se han realizado a través de un adaptador usb a Can bus que sirve para gestionar una red CAN. El adaptador es Usb to Can - compact de la casa IXXAT.

Figura 4.14: Adaptador USB-CAN de IXXAT

Este adaptador tiene el protocolo necesario para configurar una red de CAN y para gestionarla, es decir, tiene librerías que definen funciones para iniciar la tarjeta, configurar diferentes parámetros, transmitir/recibir mensajes, etc. La siguiente figura muestra el procedimiento general:

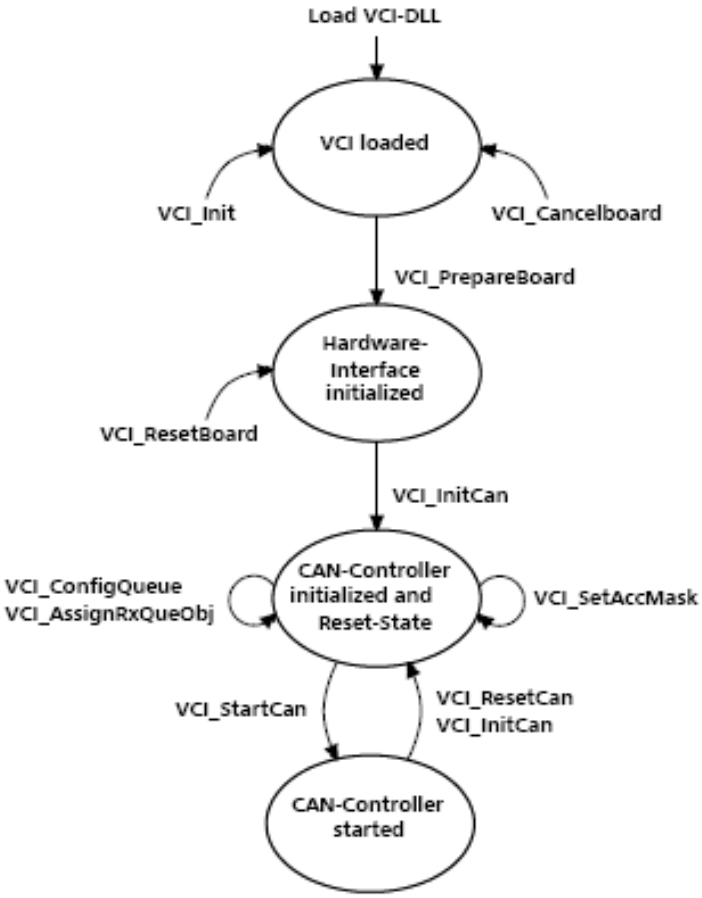


Figura 4.15: Esquema del funcionamiento del adaptador USB-CAN

Nuestra tarjeta de usb-can bus dispone de parámetros de tiempo que reflejan el tiempo en el que llegan los mensajes a la cola de mensajes de llegada, de cada nodo. Su precisión se ha establecido en 10us, para poder obtener buenas medidas.

4.5.1 Tiempos de consulta

El tiempo de consulta se calcula con la diferencia entre la llegada de un mensaje y el siguiente consecutivo. Para recibir un mensaje es necesario el envío de un mensaje de petición de información y la recepción del mensaje con dicha información. La consulta se ejecuta un número alto de veces y se calcula el tiempo medio de envío.

Debido a la conexión USB con la tarjeta que gestiona el adaptador, la velocidad máxima de comunicación con el microprocesador es 4 veces inferior que la que pudiera ser con conexión directa a una ranura PCI o miniPCI, pasando de 2,11 Gbps a sólo 480 Mbps en el caso del USB. Estas velocidades podemos considerarlas despreciables, si consideramos la velocidad del CANbus de 1Mbps.

Los mensajes que se envían tienen, excepto los mensajes de inicialización que no se consideran, 130 bits, independientemente de si se usan todos los 64 bits de parámetros o no. En el proceso de consulta el tráfico es de 2 mensajes (260 bits)

Tiempo medio calculado recepción mensaje = 320us

Ancho de banda = 812.5 Kbps muy cercano a 1Mbps

Aunque teóricamente para enviar 260 bits, el tiempo necesario es 260us, en el tiempo medido hay que incluir los tiempos de cómputo de los microprocesadores. El microprocesador de la CPU, el del adaptador USB-CAN y el del driver del motor.

4.5.2 Tramas

4.5.2.1 Cargado de puntos

Una de los métodos de control de los motores es la carga continua de puntos de su trayectoria. Se pueden realizar cargas de tramas con varios puntos, el driver los interpola según los requisitos de velocidad y tiempo, y al terminar la ejecución se realiza una lectura de la posición real y la demandada, para detectar errores y corregir desviaciones en la trayectoria, para cargar la trama siguiente correcta.

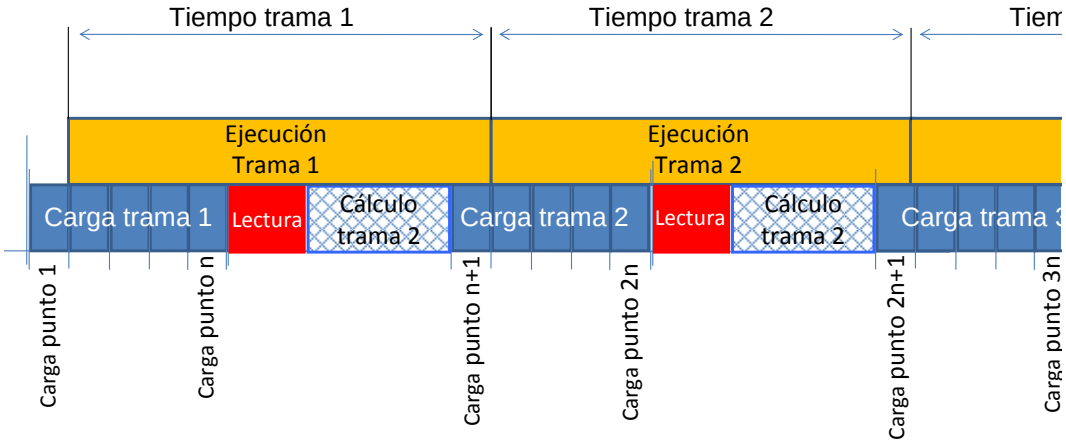


Figura 4.16: Ciclo de una carga de tramas

4.5.2.2 Tiempos de cargado

Aún no se han estimado los métodos de carga de puntos, ni el tamaño de las tramas en caso de cargar varios puntos. A continuación se estudian las limitaciones y relaciones entre los puntos cargados y los tiempos de ejecución y carga.

Se han considerado tiempos teóricos de envío de 1 mensaje de 130us. Además de la carga de una trama, se necesita mandar dos mensajes para la consulta y cargar al menos un punto de la siguiente trama, para que el movimiento sea continuo.

Por tanto en cada ciclo se envían 3 mensajes más (n-1) mensajes de carga de puntos, donde n es el número de puntos por trama.

$t_{mín\ ejecución}$= (3 + (n-1))*$t_{mensaje}$=(2+n)*130us= 260 + 130*n (us)

n_{min}=2

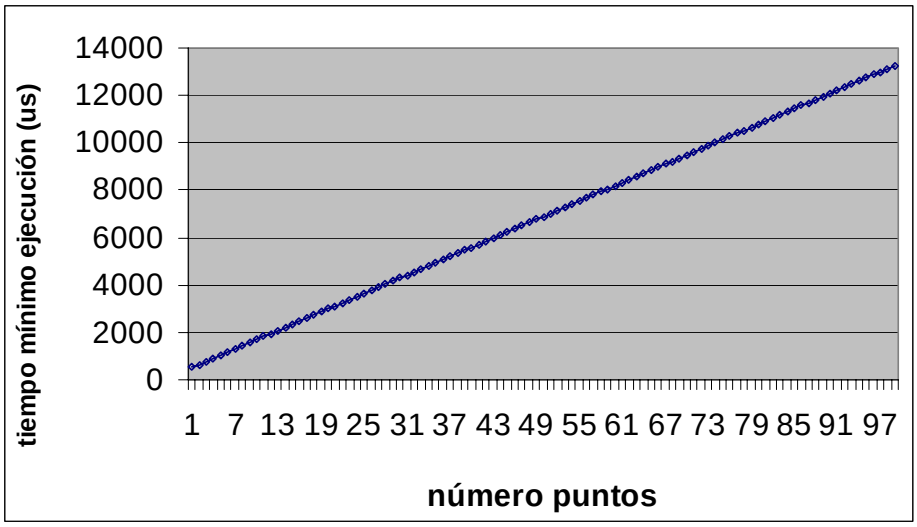


Figura 4.17: Relación entre el num. de puntos y el t. mínimo de ejecución

Se calcula el tiempo mínimo de ejecución, para disponer de tiempo de cómputo. El tiempo de ejecución debe ser la del tiempo de carga de (2+n) mensajes más el tiempo de cómputo. $t_{ejecución}$=t_{carga} (2+n) + $t_{cómputo}$

Caso particular caminata

Para el estudio de la caminata del robot, se estima la duración de un paso y los puntos cargados. Se ejecutan 175 puntos en 0.75s, en período de ejecución iguales por lo tanto de 4.28 ms.

A partir de este dato se obtiene el tiempo de cómputo disponible en cada caso. El tiempo de cómputo es el tiempo sobrante, para el cálculo y corrección de la siguiente trama. Es imprescindible disponer de tiempo suficiente para poder llevar a cabo el control de la trayectoria. El tiempo de cómputo será la diferencia entre el tiempo de ejecución de una trama de n puntos y el tiempo mínimo de ejecución de los n puntos.

$$t_{cómputo} = . t_{ejecución} - t_{carga}(2+n) = 4.28ms*n - 130us*(2+n)$$

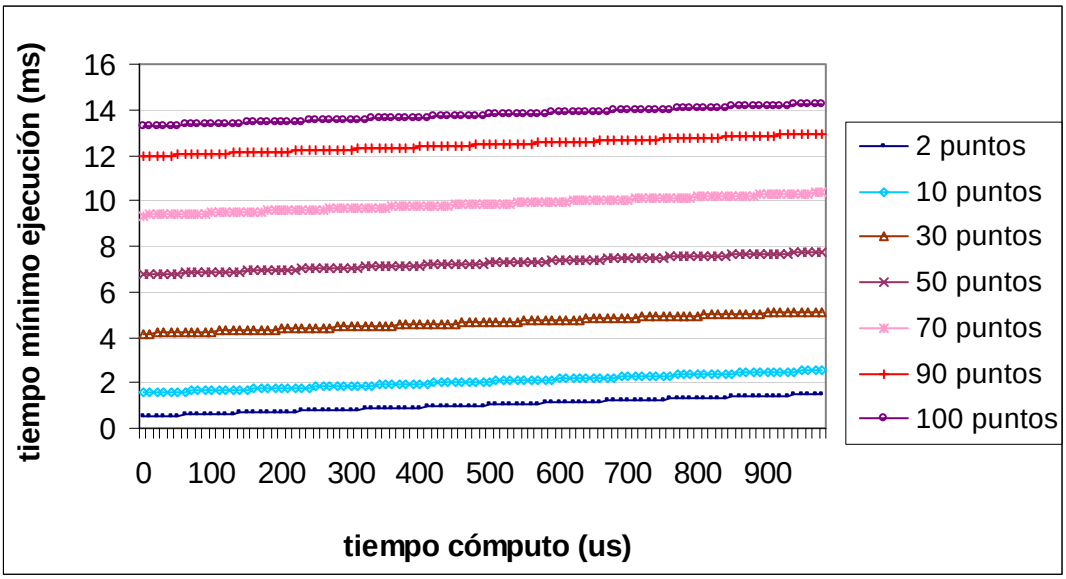


Figura 4.18: Relación tiempo de cómputo y tiempo mínimo de ejecución

4.5.3 Punto a punto

4.5.3.1 Cargado de puntos

También se pueden cargar punto a punto, es decir, realizar una lectura final de la posición después de la ejecución de cada punto, y cargar el siguiente punto, una vez corregida la trayectoria. Con este método se consigue minimizar el error, que en el caso de la trama se arrastra durante la ejecución de los puntos de la trama.

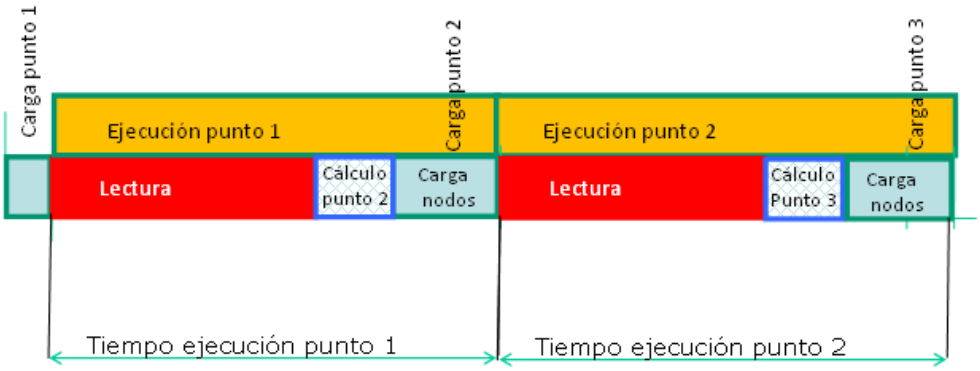


Figura 4.19: Carga de trayectorias punto a punto

Para comprobar que el driver lleva a cabo la interpolación, se han cargado cuatro puntos, correspondientes a distintas velocidades cada uno.

4.5.3.2 Tiempos de cargado

Para seguir este método, existen limitaciones en cuánto al tiempo de carga y lectura de los datos. Estos tiempos deben ser reducidos y despreciables comparado con el tiempo de ejecución. Estudiaremos el peor de los casos de la carga de un punto en los drivers.

		Tobillo Roll M01-M12		Tobillo Pich M02-M11		Rodilla Pich M03-M10		Cadera Pich M04-M09		Cadera Roll M05-M08		Cadera Yaw M06-M07	
		Max.	RMS	Max.	RMS	Max.	RMS	Max.	RMS	Max.	RMS	Max.	RMS
Torque (mNm)		788,5	242,5	684	194,5	816,5	240,5	450	153	481,5	281,5	146	60
RPM		1800		5100		6400		5100		1800		1100	
Modelo Motor	Corriente RMS (A)	273755	4,59	218009	2,60	148877	4,34	273754	3,39	273757	4,42	118755	1,56
	Par Max Adm (mNm)		967,00		1580,00		2500,00		1070,00		766,00		280,00
	Corriente Pico (A)	RE35-90W	14,91	RE40-150W	9,14	RE40-150W	14,72	RE35-90W	9,97	RE35-90W	7,56	RE25-20W	3,79
	Volt. Maximo (V)		13,75		48,31		44,03		32,58		17,01		5,73

Tabla 4.48: Datos de la arquitectura del RH-2

La máxima velocidad que alcanza un motor, es 6400 rpm en el caso del motor de la rodilla.

Según los estudios cinemáticos del robot, la duración de un paso será de 0.75s, y conlleva la carga de 175 puntos, para describir su trayectoria. A partir de estos datos, se calcula la rotación que realiza el motor en un tiempo mínimo.

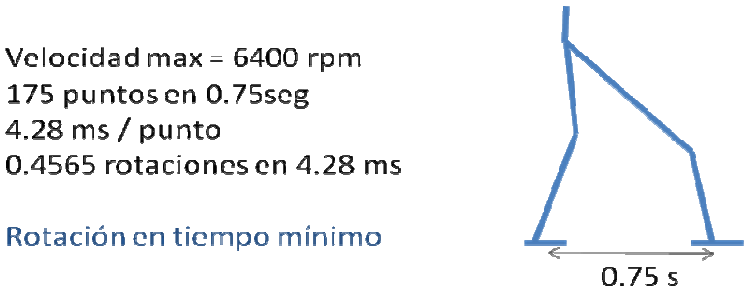


Figura 4.20: Datos para la generación de caminatas

Experimentalmente se ha obtenido una velocidad de 812 Kbps. Ya se ha comentado el retardo que introducen los tiempos de cálculo de los microprocesadores. Tampoco las condiciones y la tarjeta que controla el Bus, son las mismas que las del robot Rh-2. Por lo tanto se calculan los tiempos de envío a partir de la velocidad teórica máxima de 1Mbps.

Tiempo envío (130bits)=160us.

4.5.4 Elección configuración red CAN.

El método que se pretende utilizar para la carga de trayectorias, es punto a punto. Éste es el método que limita la velocidad de la red CAN. La tasa de envío de mensajes es mayor que el método de carga de tramas.

A continuación se estudia cuál de las dos configuraciones sugeridas es mejor en cuánto a los tiempos mínimos de ejecución. Mientras menor sea este parámetro antes se podrá cerrar el ciclo de lectura corrección y carga del siguiente punto. Para los cálculos de estos tiempos se han de tener en cuenta los mensajes para la comunicación con el driver y los mensajes para comprobar la posición real de un GDL a través del encoder absoluto.

- Mensajes driver

La comunicación con el driver tiene 2 finalidades:

- Consultar la posición actual del eje del motor. Es necesario el envío de un mensaje de petición y esperar a la recepción del mensaje respuesta con el valor de la corriente.

- Carga de punto. Se necesita enviar un mensaje con los parámetros.

La comunicación con el driver supone una tasa de 3 mensajes/ciclo. También podría ser necesario hacer una consulta del flujo de intensidad lo cuál requeriría un total de 5 mensajes/ciclo

- Mensajes encoder absoluto

La comunicación con el encoder absoluto, es necesaria, para calcular el error respecto a la posición leída directamente del encoder relativo motor. Se envía un mensaje de petición y se espera la respuesta con la información de la posición de encoder. Esta respuesta es enviada en un mensaje, con los 16 bits de datos, correspondientes a la resolución del encoder absoluto.

-La comunicación con el encoder supone una tasa de 2 mensajes/ciclo

CARGA 1 punto	LECTURA valor encoders	LECTURA Driver corriente
mensajes / driver	mensajes / encoder	mensajes / Driver
1	*2*	*2*

Tabla 4.49: Tipos de mensajes enviados

Ahora se va a calcular el número de bits a enviar para cada configuración según el número de drivers y de encoder absolutos que hay en el humanoide.

Redes	**Configuración A**	**mensajes / nodo**	**bits**
Drivers pierna izq.	6	3	2340
Drivers pierna der.	6	3	2340
Drivers brazos	12	3	4680
Encoders absolutos	24	2	6240
Nodos	48		

Tabla 4.50: Mensajes en red CAN durante un ciclo. Configuración A.

Redes	Configuración B	mensajes / driver	mensajes / encoder	bits
Driver - encoders pierna izq	12	3	2	3900
Driver - encoders pierna der	12	3	2	3900
Driver - encoders brazo izq	12	3	2	3900
Driver - encoders brazo der	12	3	2	3900
Nodos	48			

Tabla 4.51: Mensajes en red CAN durante un ciclo. Configuración B.

A partir de estos datos se calcula cuál sería la velocidad del bus necesaria, para cumplir los tiempos de ejecución. Estos tiempos de ejecución en el caso de la trayectoria de la caminata son de 4.28 ms.

Por lo tanto para comprobar la viabilidad de un tipo u otro de configuración, es necesario asegurar que el tiempo total necesario para cerrar el ciclo de control de la posición de las articulaciones es menor que el tiempo de ejecución.

Los datos de la gráfica se adjuntan en el anexo, página 196.

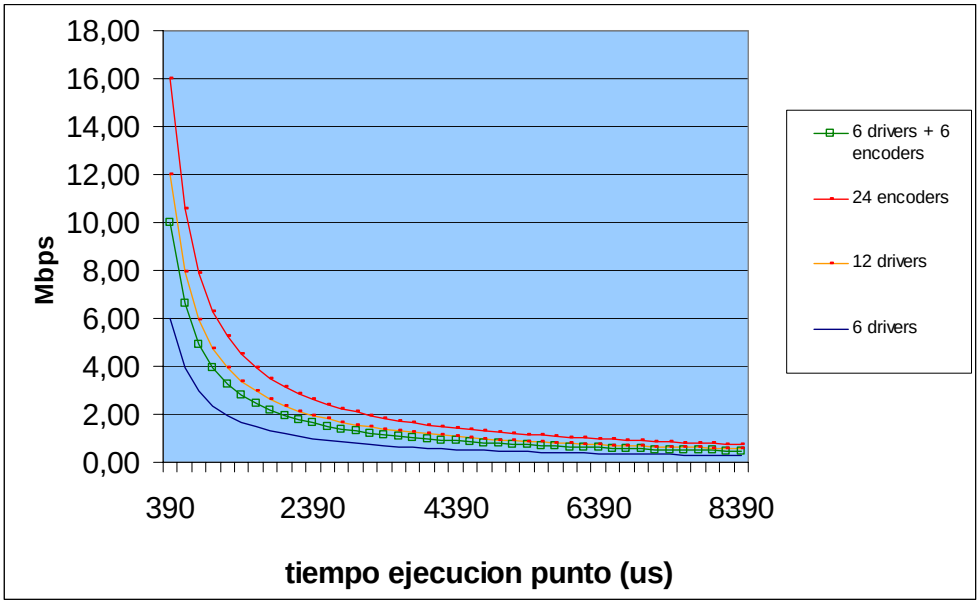


Figura 4.21: Gráfica velocidad CAN bus requerida en función del tiempo de ejecución

Nota: Los datos usados para generar esta tabla se encuentran el anexo 2.

- **Configuración A**

Para cerrar el ciclo de control de posición de las articulaciones, es necesario completar el envío/recepción de todos los bits en cada red. La red que limita el tiempo de ejecución es la red 4, formada por 24 encoders. El tiempo necesario, para enviar los 6390 bits a una velocidad de 1Mbps, es de 6.39ms. Por lo tanto considerando tiempos de ejecución de 4.28 ms < 6.39 ms, este tipo de arquitectura no es viable.

- **Configuración B**

Este tipo de configuración reparte de manera uniforme los mensajes enviados entre las cuatro redes. El tiempo para cerrar el ciclo y enviar los mensajes en cada red es el mismo.

El tiempo ejecución de 4.28ms > 3.99ms, nos asegura que el ciclo de control se completa en tiempo suficiente. Por lo tanto ésta será la configuración elegida para las redes CAN.

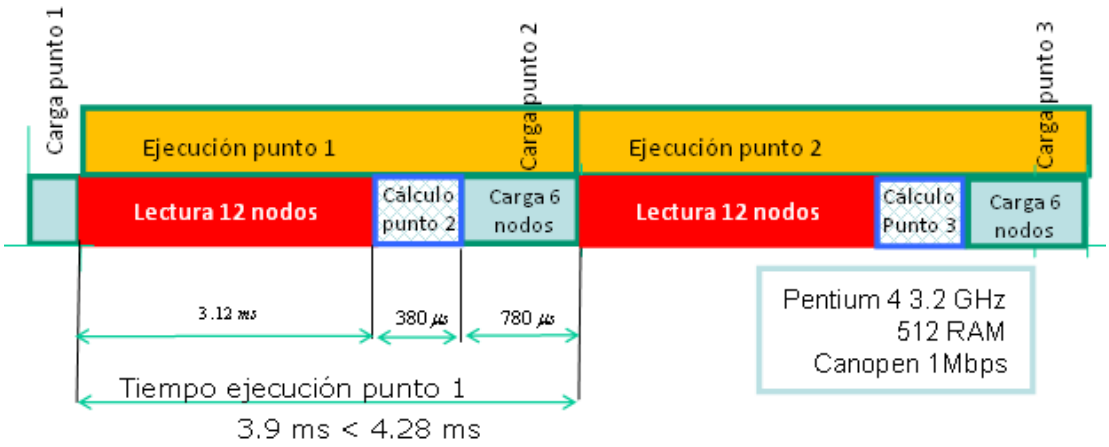


Figura 4.22: Tiempos de envío para la configuración B

4.6 Simulación del cargado de puntos

El apartado anterior se centra en el cálculo de los tiempos de transmisión, tiempo de ciclo, etc. Este apartado pretende centrarse en los mensajes a enviar para generar los dos tipos de envío (punto a punto y por tramas). Aunque puedan parecer distintos, solo se diferencian en el número de puntos que se cargan en cada ciclo, por lo que el esquema de mensajes es muy similar:

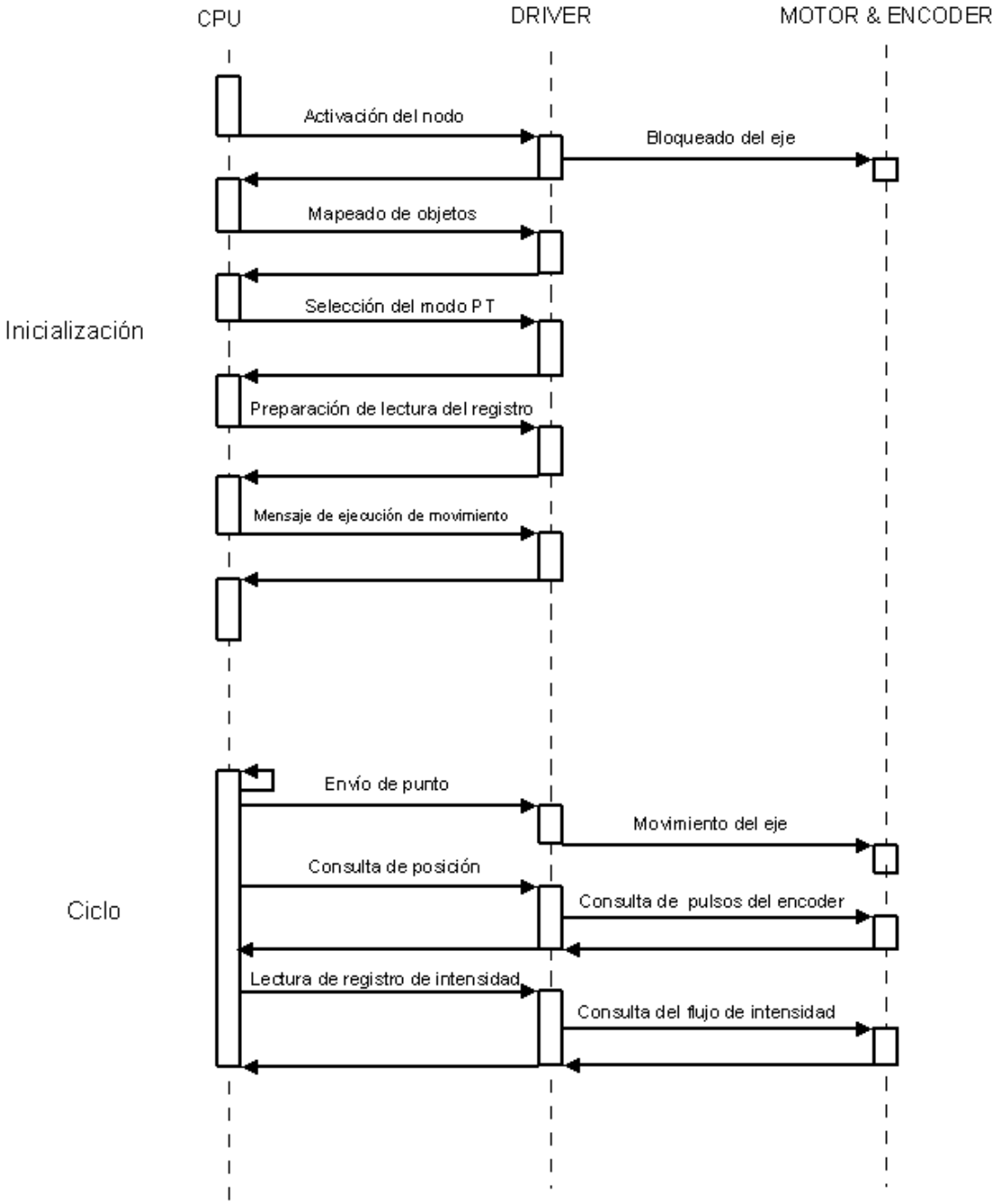


Figura 4.23: Diagrama del proceso de envío de mensajes al driver

4.6.1 Proceso de inicialización:

Esta primera parte se ha de realizar al principio para cada uno de los nodos, así cada nodo queda activo y configurado para recibir los puntos y las consultas de posición o intensidad que se realicen. Como el procedimiento es idéntico para ambos tipos de envío no se va a escindir en uno para cada configuración. En este ejemplo todo el proceso va a estar referido al nodo 6, por ello para hacer lo mismo con un nodo de id distinto basta modificar el COB-ID del mensaje. Por ejemplo si el COB-ID de un mensaje de este ejemplo es 606 y el id del nodo es 25 (19h en exadecimal), el COB-ID será 619.

1) Iniciar el nodo 6 mandando un mensaje NMT

COB-ID	Datos
0	01 06

2) Se cambia el estado del nodo de encendido deshabilitado a preparado para encenderse (ver objeto 6041h)

COB-ID	Datos
206	06 00

3) Se cambia el estado del driver a encendido

COB-ID	Datos
206	07 00

4) Se cambia el estado a operación permitida vía PDO

COB-ID	Datos
206	0F 00

5) Se desactiva el RPDO2: se escribe '0' en el objeto 1601h, subíndice 00h

COB-ID	Control de flujo de datos	Índice de objeto	Subíndice	Datos
606	2F	01 16	00	00 00 00 00

6) Se mapean lo nuevos objetos:

a) En el objeto 1601h, subíndice 1, se graba la descripción del subíndice 1 del objeto Registro de datos de interpolación (60C1h).

COB-ID	Control de flujo de datos	Índice de objeto	Subíndice	Datos
606	23	01 16	01	20 01 C1 60

b) En el objeto 1601h, subíndice 2, se graba la descripción del subíndice 2 del objeto Registro de datos de interpolación.

COB-ID	Control de flujo de datos	Índice de objeto	Subíndice	Datos
606	23	01 16	02	20 02 C1 60

7) Se habilita el RPDO2: se escribe 2 en el objeto 1601h, subíndice 00h

COB-ID	Control de flujo de datos	Índice de objeto	Subíndice	Datos
606	2F	01 16	00	02 00 00 00

8) Se selecciona el Modo de Posición Interpolada (en el objeto 6060h, valor 07h en el primer byte de datos)

COB-ID	Control de flujo de datos	Índice de objeto	Subíndice	Datos
606	2F	60 60	00	07 00 00 00

9) Selección del submodo PT (se accede al objeto 60C0h y se le da el valor FFFFh en 16-bit.

COB-ID	Control de flujo de datos	Índice de objeto	Subíndice	Datos
606	2B	C0 60	00	FF FF 00 00

10) Se configura la longitud del buffer (objeto 2074h)

COB-ID	Control de flujo de datos	Índice de objeto	Subíndice	Datos
606	2B	74 20	00	00 0C 00 00

11) Se edita la posición inicial en la que se encuentra el motor. Esto unicamente sirve para modificar el punto de referencia a la hora de interpolar los puntos. Únicamente tiene utilidad para movimientos de tipo absoluto. En nuestro caso ponemos 0 rotaciones:

COB-ID	Control de flujo de datos	Índice de objeto	Subíndice	Datos
606	23	79 20	00	00 00 00 00

12) Ahora hay que configurar el objeto Lectura/Escritura del Registro de Configuración (índice 2064h) para que cuando se ejecute una lectura del valor de la intensidad, esté correctamente definida la dirección de memoria desde la que coger el valor.

COB-ID	Control de flujo de datos	Índice de objeto	Subíndice	Datos
606	23	64 20	00	85 00 30 02

13) Se inicia un movimiento absoluto para tener al driver preparado para recibir los puntos. Este es el último mensaje que hay que mandar dentro de la parte de inicio del driver.

COB-ID	Datos
206	1F 00

Nota: para que el modo fuese relativo habría que enviar el mensaje 5F 00 y tener en cuenta que la información a enviar los puntos es distinta ya que se envían incrementos de la posición anterior y no la posición a alcanzar

4.6.2 Mensajes de los ciclos

Una vez inicializado el nodo y habiendo quedado éste a la espera de recibir los puntos, ya es cuando quedan diferenciados el proceso de enviar punto a punto o por tramas.

4.6.2.1 Ciclo del envío punto a punto

1) Se envía el primer punto. Éste ha de tener como valor de tiempo la duración del tiempo de ciclo. Se ha ordenado que vaya a la posición 20 rotaciones (20x2000= 40000 = 9C40h) en 2 segundos (2000 = 7D0h)

COB-ID	Datos
306	40 9C 55 00 D0 07 00 00

2) A mitad del ciclo se manda un mensaje de consulta de la posición actual

COB-ID	Control de flujo de datos	Índice de objeto	Subíndice	Datos
606	40	64 60	00	00 00 00 00

3) Si fuese necesario, se puede mandar un mensaje de consulta para conocer el consumo de intensidad que tiene el motor. Este mensaje se puede mandar o entre el mensaje de consulta de la posición y el siguiente punto (durante el proceso de recálculo del punto) o entre el envió del punto y el mensaje de consulta de la posición

COB-ID	Control de flujo de datos	Índice de objeto	Subíndice	Datos
606	40	66 20	00	00 00 00 00

4) Se envía el segundo punto. El punto se ha de enviar antes de que acabe el ciclo del punto anterior para que el movimiento sea continuado. Se ha ordenado que el motor vaya a la posición 0 rotaciones en 2 segundos. Hay que tener en cuenta el incremento del valor del contador sabiendo que son los 7 MSB del primer byte

COB-ID	Datos
306	00 00 00 00 D0 07 00 02

5) A mitad de este ciclo se vuelve a enviar un mensaje de consulta de la posición actual

COB-ID	Control de flujo de datos	Índice de objeto	Subíndice	Datos
606	40	64 60	00	00 00 00 00

El perfil generado es el siguiente:

Perfil de posición Perfil de velocidad

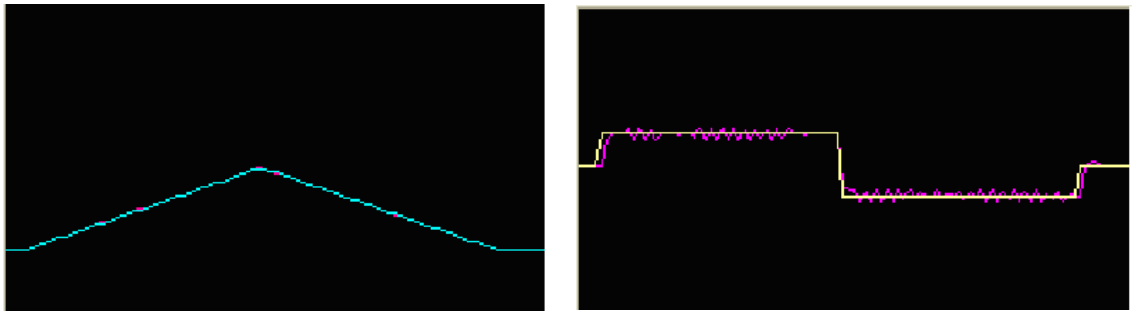


Tabla 4.52: Perfil obtenido en la simulación punto a punto

Los datos devueltos por las consultas de posición son:

- Primera consulta: 10,1 rotaciones
- Segunda consulta: 10 rotaciones

El valor devuelto por la consulta de la intensidad es: 205 mA*

**Nota: el valor de intensidad puede que no sea exacto*

4.6.2.2 Ciclo del envío por tramas

1) Se envía la primera trama compuesta por tres puntos. Se va ha hacer que el driver vaya de la posición inicial a 20 rotaciones, luego a 30 y retorne a 20 de nuevo:

COB-ID	Datos
306	40 9C 00 00 D0 07 00 00

COB-ID	Datos
306	60 EA 00 00 D0 07 00 02

COB-ID	Datos
306	40 9C 00 00 D0 07 00 04

2) A mitad del ciclo o cuando se desee, se manda un mensaje de consulta de la posición actual

COB-ID	Control de flujo de datos	Índice de objeto	Subíndice	Datos
606	40	64 60	00	00 00 00 00

3) Si fuese necesario, se puede mandar un mensaje de consulta para conocer el consumo de intensidad que tiene el motor. Este mensaje se puede enviar entre el mensaje de consulta de la posición y el siguiente punto (durante el proceso de recálculo del punto) o bien entre el envió del punto y el mensaje de consulta de la posición

COB-ID	Control de flujo de datos	Índice de objeto	Subíndice	Datos
606	40	66 20	00	00 00 00 00

4) Se envía la siguiente trama que hará que el driver vaya a las posiciones 50 rotaciones, 25 rotaciones y acabe en la posición inicial:

COB-ID	Datos
306	A0 86 01 00 D0 07 00 06

COB-ID	Datos
306	50 C3 00 00 D0 07 00 08

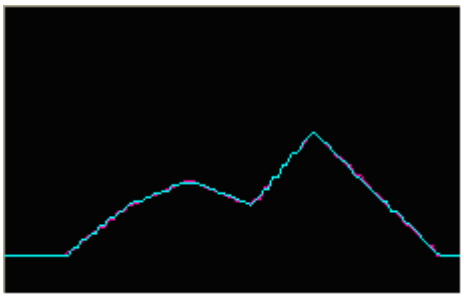
COB-ID	Datos
306	00 00 00 00 D0 07 00 0A

5) Se vuelve a enviar un mensaje de consulta de la posición actual cuando sea requerido

COB-ID	Control de flujo de datos	Índice de objeto	Subíndice	Datos
606	40	64 60	00	00 00 00 00

El perfil generado es el siguiente:

Perfil de posición



Perfil de velocidad

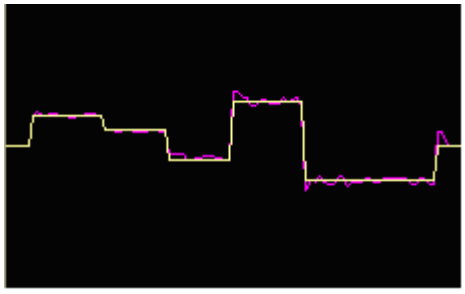


Tabla 4.54: Perfiles obtenidos en la simulación del envío por tramas

Los datos devueltos por las consultas de posición son:

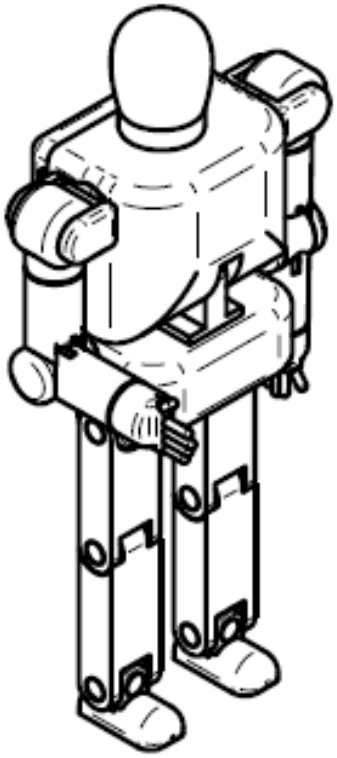
- Primera consulta: 17,7 rotaciones
- Segunda consulta: 25,4 rotaciones

El valor devuelto por la consulta de la intensidad es: 343 mA*

**Nota: el valor de intensidad puede que no sea exacto*

4.6.2.3 Mensajes de las configuraciones de CAN

En el apartado 4.5 se analizaron dos tipos de configuraciones según la forma en que se agrupasen los encoders o los drivers de cada pierna, brazo, etc. Para generar cada configuración, el único procedimiento que hay que realizar es agrupar mensajes del mismo tipo (por ejemplo cargar un punto) y enviarlos a varios nodos de forma simultánea. Como los mensajes a enviar son los mismos que en apartados anteriores, no se va a proceder a describir como se desarrolla el envío de mensajes. Sólo indicar que para este proceso habría antes que inicializar cada uno de los drivers del humanoide como se indicó en el apartado 4.6.1 ya que seguramente los nodos de los encoders absolutos no necesiten este procedimiento.



Capítulo 5: Conclusiones

Conclusiones de este proyecto y trabajos futuros a realizar.

5.1 CONCLUSIONES

La robótica es un campo en continua expansión en el que intervienen casi todas las ramas de la ingeniería: cinemática, dinámica, mecánica, regulación y control, electrónica, etc. El desarrollo de humanoides está haciendo que países como Japón sean pioneros en un campo aun por explotar a nivel mundial.

Según va pasando el tiempo, están llegando al mercado nuevos prototipos con increíbles avances tecnológicos. Robots como el ASIMO, EL HRP-3 o el QRIO son prueba de ello. Es probable que a medio plazo surja una gama de robots humanoides capaces de trabajar de forma estable en entornos industriales o de asistencia a personas. Es por ello necesario una competencia comercial para que el desarrollo de nuevos proyectos de investigación se produzca de una manera más eficiente y obtener más avances en menos tiempo.

El desarrollo del nuevo RH-2 en la Universidad Carlos III debe ser una apuesta segura y basada en un sólido estudio previo. Este prototipo permitirá estar a la altura de los humanoides actuales y dar un paso más en este proceso de continuas mejoras. En todo momento se ha tenido el objetivo en esta primera fase, de crear la parte caminante del robot. Elegir la forma correcta par controlar cada uno de los GDL de la forma más eficiente posible encamina al éxito del proyecto.

Ha sido difícil documentarse y obtener conclusiones a partir de los escasos datos que facilitan los centros de investigación, en concreto los japoneses. Se ha hecho un estudio sobre la evolución de los sistemas de control y se han analizado los sistemas de control de algunos de los humanoides más importantes. Se han analizado los proyectos anteriores desarrollados en este departamento.

Para el prototipo actual (RH-2) se ha partido de los robots previamente creados y se ha conseguido diseñar y elegir componentes, tanto estructurales como del sistema de control, consiguiendo así un sistema preparado para realizar movimientos y tareas más complejos.

Este proyecto en concreto, se inicia con el análisis del driver HARMONICA de ELMO que usaban los prototipos anteriores, y a partir de él se inició una búsqueda para encontrar un sustituto. Una vez encontrado el controlador ISCM8005 y observando que puede ser un candidato, se inicia una comparación entre ambos controladores, de lo que se concluye que la tarjeta ISCM, se adapta mejor a los requerimientos del RH-2 y ofrece recursos que pueden ser útiles a medio plazo.

A raíz de esto, se hace una descripción exhaustiva de todas sus propiedades y se comienza a realizar pruebas a través del programa EasyMotion para recopilar información y poder así generar una base sobre la que crear el manual de usuario del driver. En este primer proceso se hacen pruebas con distintos motores y encoders. Con esto se consigue generar documentación sobre los distintos conexionados y procedimientos necesarios para el montaje del driver. Un segundo paso fue probar cada modo de funcionamiento.

Una vez adquiridos ciertos conocimientos sobre su funcionamiento, se procedió a incluir el driver dentro de una red de de comunicaciones de CANbus. Por esta razón se recopila información sobre este bus de datos y del protocolo que va a utilizarse. A partir de ahí, se intentan adaptar los modos de control estudiados previamente a CANOpen y se describe la secuencia de mensajes a enviar de cada uno. También se realizan pruebas para analizar los tiempos de transmisión y averiguar qué tipo de configuración de red ofrece más prestaciones de velocidad. A modo de simulación del envió de mensajes que necesitará el control de motores de cada GDL de nuestro humanoide, se hace un resumen de todos aquellos a utilizar desde la inicialización hasta los envíos cíclicos.

Una vez finalizado el proyecto, se puede afirmar que se ha llegado a las siguientes conclusiones u objetivos:

1. El driver ISCM8005 es adecuado para controlar los GDL del robot RH-2.

2. Se ha integrado el driver en una red CAN y se ha comprobado que desarrolla de forma correcta sus actividades (modos de control, llamadas a funciones, etc).

3. Se ha obtenido la configuración de red de CAN más adecuada para cerrar los ciclos en el menor tiempo posible.

4. Se ha desarrollado el procedimiento que probablemente se use para el control de los GDL del humanoide.

5.2 TRABAJOS FUTUROS

Es necesario realizar el montaje de una red semejante a la definitiva del bus de datos CAN y que el programa de control de ésta sea desarrollado a conciencia para minimizar retardos durante la transmisión de mensajes. Esto unido a la inserción de varios drivers al bus, haría posible la realización de pruebas realistas sobre tiempos de transmisión y la viabilidad de una u otra configuración.

Se sugiere probar el envío de mensajes broadcast de consulta a los nodos de cada red, para leer las posiciones de los encoder relativos y absolutos. Esto supondría una considerable disminución de tráfico en cada red. Se mandaría 1 mensaje en vez de 12. Para permitir que los nodos de los PIC (6 en cada red) acepten los mensajes que queramos, es necesario acceder a sus registros y cambiar los filtros de aceptación de mensajes (Filter Acceptance Message). Para el caso de los controladores ISCM, se podría estudiar como cambiar estos registros, por ejemplo accediendo al chip que integra con el protozoo CAN bus descrito.

Una posible mejora, sería utilizar los objetos CAN de llamada a funciones y al programa TML que se pueden almacenar en la memoria del driver. En dichas funciones y programa se pueden almacenar procesos que se ejecuten de forma continuada. Así se optimizaría el uso del bus ya que eliminaría trasmisiones a través de él. Esta es una mejora que en mi opinión merece la pena ser estudiada a fondo y se plantee el aplicarla en el control del humanoide.

Otra mejora consiste en utilizar otros objetos CAN para controlar y configurar el driver, pero accediendo a ellos a través de PDO’s. Para ello se propone el mapeo dinámico de todos aquellos objetos que se usen frecuentemente. Accediendo a los objetos mediante PDOs se reduce el tiempo de envío y se optimiza el bus para el uso en tiempo real, aunque se cancela la confirmación (ACK). Por lo tanto hay que decidir si prima la seguridad en el envío de datos, o la velocidad.

Otro trabajo futuro podría ser el analizar modos de control como EGEAR ó ECAM que no han sido probados durante la realización de este proyecto pues en un primer momento parecieron carecer de interés, pero que quizás en un estudio a posteriori se les pueda sacar alguna utilidad.

Bibliografía

Recursos de información utilizados en
la redacción de este proyecto

Manuales:

- Minimon Manual
- Virtual CAN Interface V2 Programmers Manual
- Harmonica Software Manual
- ISCM8005 Technical Reference
- MotionChip II TML Programming
- CANopen Programming
- ISCMxx05 Starter Kit User Manual
- IO ISCM User Manual
- MotionChip II Configuration Setup
- MotionChip II TML Programming

Páginas Web:

Se incluyen algunas páginas web que en su momento se anotaron. En realidad muchas más páginas han sido consultadas.

- http://www.microsiervos.com/archivo/tecnologia/robots-en-el-mundo.html
- http://www.technsoftmotion.com
- http://www.elmomc.com
- http://www.faulhaber.com
- http://www.maxonmotor.es
- http://www.can-cia.org
- http://asimo.honda.com
- http://www.sony.nt/SonyInfo/QRIO/
- http://www.toyota.co.jp/en/special/robot/
- http://www.fiveco.com/
- http://ww.wikipedia.es
- http://www.androidworld.com
- http://www.embeddedsys.com

Proyectos fin de carrera:

Universidad Carlos III Madrid

- Desarrollo de la arquitectura hardware de control de ejes del robot humanoide RH-0

  Autor: Miguel Ángel Rodríguez Sánchez
- Implementación de las comunicaciones mediante CANbus en el robot ASIBOT_v1.5

  Autor: Víctor Placer de Miguel
- Diseño e implementación de hardware de instrumentación, sistema de control de temperatura y sistema de sincronismo de ejes para el robot humanoide RH-1

  Autor: Javier Pérez García-Silvestre
- Arquitectura del sistema de control de movimientos del robot humanoide RH-cero

  Autor: Dmitry Kaynov

Universidad de Salamanca

- CANopen

  Autora: Raquel Sánchez Díaz

Universidad de la República Oriental del Uruguay

- Construcción de Robots Bípedos

  Autores: Damián Lezama y Alexander Sklar

Artículos y presentaciones:

- Técnicas de inteligencia artificial en la construcción de robots móviles autónomos

  José Andrés Vicente Lober
- CAN/CANopen introduction

  Andreas Frölich
- Control de motores eléctricos
- Digital Position Control for Analog Servos

Sven Behnke Michael Schreiber

- Historia del control automático

Libros:

- Robótica industrial : fundamentos y aplicaciones
  Rentería, Arantxa , McGraw-Hill
- Fundamentos de robótica
  Barrientos Cruz, Antonio, McGraw-Hill

Anexos

Códigos fuente, hojas características y otros anexos

Anexo 1: Programa de CANbus

Este programa ha sido desarrollado para mandar puntos por tramas o punto a punto. En los ciclos se realizan consultas de la posición y del valor del flujo de la intensidad.

```
/*****************************************************************************
**    IXXAT Automation GmbH
*****************************************************************************
**
**      File    : vci_demo.cpp
**      Summary : Demo of the 'Vitual-CAN-Interface'
**                Demonstrates the following features
**                 - Hardware selection dialog
**                 - board initialisation
**                 - creation of queues and buffers
**                 - transmit/receive can messages
**
**      Version : 2.0
**      Date    : 11.8.00
**  Compiler    : 22222222VC++ 6.0
**      Author  : A. NAVARRO
*****************************************************************************
**    all rights reserved
*****************************************************************************/

/*****************************************************************************
**    compiler-instructions
*****************************************************************************/

/*****************************************************************************
**    include-files
*****************************************************************************/
#include <windows.h>

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <time.h>
#include <string.h>

#include "XatXXReg.h"       // IXXAT Registry access
#include "Xatdynl.h"       // dynamically linking of XAT10Reg.DLL
#include "VCI2.h"

/*****************************************************************************
**    global variables
*****************************************************************************/
DECLARE_FUNCTION_POINTER_EXT(XAT_SelectHardware);

/*****************************************************************************
**    static constants, types, macros, variables
*****************************************************************************/

#define WAIT_TIME  2000 // wait to display messages

#define  CAN_NUM            0
#define  ANZ_BUFFERS         10
#define  TX_CYCLE_TIME  (1000/55)   /* ms */
```

```
static UINT16 BoardHdl;
static UINT16 TxQueHdl;
static UINT16 RxQueHdl2;
static UINT16 BufHdl[ANZ_BUFFERS];

/*****************************************************************************
**    static function-prototypes
*****************************************************************************/
void init_can(void);
void init_board(void);
void carga_puntos(void);
void carga_tramas(void);
void receive_buffer_data(void);
void receive_queuedata( UINT16 que_hdl, UINT16 count, VCI_CAN_OBJ FAR * p_obj);
void exception_handler( VCI_FUNC_NUM func_num, INT32 err_code, UINT16 ext_err, char
*err_str);
void message_handler  ( char *msg_str );

void end_app();

/*****************************************************************************
**    global functions
*****************************************************************************/

/*****************************************************************************
**
**    Function    : main
**
**    Description : main function
**    Parameters  : -
**
**    Returnvalues: -
**
*****************************************************************************/
void main (void)
{

  printf(" Initialize Board...\n");
                          /* Initialize Board*/
  init_board();
  printf(" Initialize Board..... OK !\n\n");
  printf(" Initialize CAN...\n");
  init_can();                 /* Initialize CAN-Controller */
  printf(" Initialize CAN..... OK !\n\n");

  while(1)
  {
   int tecla;
  printf("\ncargar tramas (pulsa a)");
  printf("\ncargar puntos (pulsa b)");
  tecla=getch();
  switch(tecla)
  { case'a': carga_tramas(); break;
    case'b': carga_puntos(); break;
    default: printf("\npulsa tecla correcta"); break;
  }
}
}
/*****************************************************************************
**    static functions
```

```
*****************************************************************************/

/*****************************************************************************
**
**    Function    : init_board
**
**    Description : Loads the XAT registry DLL and pos up the
**                  board selection dialog.
**                  After the user has selected a board it is tried to
**                  initialize this board.
**    Parameters  : -
**
**    Returnvalues: -
**
*****************************************************************************/
void init_board( )
{
  int i_test;
  XAT_BoardCFG sConfig ;

  /*
  **  load hardware selection DLL dynamically
  */
  if (!LoadXatLib())
  {
    printf("Cannot load XAT10Reg.DLL");
    exit(-1);
  }

  /*
  **  copy function pointers for DLL functions to global variables
  */
  if (MapXATDialogs() < 0)
  {
    printf("Cannot map functions in XAT10Reg.DLL");
    exit(-1);
  }

  /*
  **  show hardware selection dialog
  */
  if ( DYNCALL(XAT_SelectHardware)( NULL, &sConfig ))
  {
    /*
    **  prepare choosen board for further configuration
    */
    i_test = VCI2_PrepareBoard( sConfig.board_type      // board type
                          , sConfig.board_no        // unique board index
                          , NULL                  // pointer to buffer for additional info
                          , 0                     // length of additional info buffer
                          , message_handler         // pointer to msg-callbackhandler
                          , receive_queuedata       // pointer to receive-callbackhandler
                          , exception_handler);     // pointer to exception-callbackhandler
    if(i_test < 0)
      end_app();

    // extract boardhandle from return value
    BoardHdl = (UINT16) i_test;
  }
```

```
 /*
 **  unmap function pointers for DLL functions
 */
 UnmapXATDialogs();

 /*
 **  unload hardware selection DLL
 */
 FreeXatLib();
}


/*****************************************************************************
**
**    Function    : init_can
**
**    Description : Initialize CAN controller and create queues and
**                  buffers
**    Parameters  : -
**
**    Returnvalues: -
**
*****************************************************************************/
void init_can(void)
{
 int ret;
 int i;

 /*
 **  initialize CAN-Controller
 */

  ret = VCI_InitCan(BoardHdl, CAN_NUM, 0x00,0x14, VCI_11B);

 /*
 **  definition of Acceptance-Mask (define to receive all IDs)
 */
 ret = VCI_SetAccMask(BoardHdl, CAN_NUM, 0x0UL, 0x0UL);

 /*
 **  definition of Transmit Queue
 */
 ret = VCI_ConfigQueue(BoardHdl, CAN_NUM, VCI_TX_QUE, 100 , 0, 0, 0,  &TxQueHdl);

 /*
 **  definition of Receive-Buffers (Identifier 0x100 - (0x100+ANZ_BUFFERS)).
 */
 for(i = 0; i < ANZ_BUFFERS; i++)
 {
  ret = VCI_ConfigBuffer( BoardHdl
                        , CAN_NUM
                        , VCI_RX_BUF
                        , (UINT32) 0x586UL+i
                        , &BufHdl[i]);
 }

 /*
 **  definition of Receive Queue (interrupt mode)
 */
 ret = VCI_ConfigQueue(BoardHdl, CAN_NUM, VCI_RX_QUE, 50, 1, 0, 100,  &RxQueHdl2);
```

```
  /*
  **  assign the ID 200 to the Receive Queue
  */
  ret = VCI_AssignRxQueObj(BoardHdl, RxQueHdl2 ,VCI_ACCEPT, 0x586, 0xFFFF) ;

  /*
  **  And now start the CAN
  */
  ret = VCI_StartCan(BoardHdl, CAN_NUM);
}

/*****************************************************************************
**
**    Function    : carga_trama
**
**    Description : Función para cargar tramas al driver con id 6
**    Parameters  : -
**
**    Returnvalues: -
**
*****************************************************************************/
void carga_trama(void)
{
int tecla;
  static clock_t time = TX_CYCLE_TIME;
  static UINT8 buf0[2] = {0x01,0x06};
  static UINT8 buf1[2] = {0x06,0x00};
  static UINT8 buf2[2] = {0x07,0x00};
  static UINT8 buf3[2] = {0x0F,0x00}; //set to zero (enable)
  static UINT8 buf4[8] = {0x2F,0x01,0x16,0x00,0x00,0x00,0x00,0x00} ;
  static UINT8 buf5[8] = {0x23,0x01,0x16,0x01,0x20,0x01,0xC1,0x60} ;
  static UINT8 buf6[8] = {0x23,0x01,0x16,0x02,0x20,0x02,0xC1,0x60};
  static UINT8 buf7[8] = {0x2F,0x01,0x16,0x00,0x02,0x00,0x00,0x00};
  static UINT8 buf8[8] = {0x2F,0x60,0x60,0x00,0x07,0x00,0x00,0x00};
  static UINT8 buf9[8] = {0x2E,0xC0,0x60,0x00,0x00,0x00,0x00,0x00};
  static UINT8 buf10[8] = {0x2B,0x74,0x20,0x00,0x00,0x0C,0x00,0x00};
  static UINT8 buf11[8] = {0x23,0x79,0x20,0x00,0x00,0x00,0x00,0x00}; //posición inicial
  static UINT8 buf12[8] = {0x40,0x9c,0x00,0x00,0xd0,0x07,0x00,0x00}; //primer punto
  static UINT8 buf13[8] = {0x00,0x00,0x00,0x00,0xd0,0x07,0x00,0x02}; //segundo punto
  static UINT8 buf14[8] = {0x40,0x9c,0x00,0x00,0xd0,0x07,0x00,0x04}; //tercero
  static UINT8 buf15[8] = {0x00,0x00,0x00,0x00,0xd0,0x07,0x00,0x06}; //cuarto
  static UINT8 buf22[8] = {0x40,0x9c,0x00,0x00,0xd0,0x07,0x00,0x08}; //quito
  static UINT8 buf23[8] = {0x00,0x00,0x00,0x00,0xd0,0x07,0x00,0x0A}; //sexto
  static UINT8 buf16[8] = {0x23,0x64,0x20,0x00,0x85,0x00,0x30,0x08}; //configuración
lector registro
  static UINT8 buf17[8] = {0x40,0x66,0x20,0x00,0x00,0x00,0x00,0x00}; //lectura registro
  static UINT8 buf19[2] = {0x1F,0x00};
  static UINT8 buf20[8] = {0x40,0x64,0x60,0x00,0x00,0x00,0x00,0x00}; //Posicion actual
  static UINT8 buf21[8] = {0x40,0x40,0x60,0x00,0x00,0x00,0x00,0x00};


while(1){


        printf("1 - start remote node - ready to switch on - switch on\n");
    printf("2 - modo pt\n");
    printf("3 - ejecutar\n");
        printf("4 - trama 1\n");
        printf("5 - trama 2\n");
```

```
 tecla=getch();

switch(tecla){

case'1':

if (VCI_TransmitObj(BoardHdl, TxQueHdl, 0x0   , 2 , buf0) == VCI_OK)
printf("started remote node\n");


 if (VCI_TransmitObj(BoardHdl, TxQueHdl, 0x206 , 2 , buf1 ) == VCI_OK)
                printf("ready to switch on\n");


      if (VCI_TransmitObj(BoardHdl, TxQueHdl, 0x206 , 2 , buf2 ) == VCI_OK)
                printf("switched on\n");
       Sleep(2000);

if (VCI_TransmitObj(BoardHdl, TxQueHdl, 0x206 , 2 , buf3) == VCI_OK)
                printf("enabled\n");

      break;

      case'2':

if (VCI_TransmitObj(BoardHdl, TxQueHdl, 0x606 , 8 , buf4) == VCI_OK)
                printf("set mode\n");

      if (VCI_TransmitObj(BoardHdl, TxQueHdl, 0x606 , 8 , buf5) == VCI_OK)
                printf("set vueltas\n");

      if (VCI_TransmitObj(BoardHdl, TxQueHdl, 0x606 , 8 , buf6) == VCI_OK)
                printf("set speed\n");

      if (VCI_TransmitObj(BoardHdl, TxQueHdl, 0x606 , 8 , buf7) == VCI_OK)
                printf("executing...\n");


      if (VCI_TransmitObj(BoardHdl, TxQueHdl, 0x606 , 8 , buf8) == VCI_OK)
                printf("reciving demand position value...\n");


      if (VCI_TransmitObj(BoardHdl, TxQueHdl, 0x606 , 8 , buf10) == VCI_OK)
                printf("consumo\n");

      if (VCI_TransmitObj(BoardHdl, TxQueHdl, 0x606 , 8 , buf16) == VCI_OK)
                printf("configurado lector registro\n");


      break;

      case '3':

      if (VCI_TransmitObj(BoardHdl, TxQueHdl, 0x606 , 8 , buf11) == VCI_OK)
                printf("posicion inicial...\n");
```

```
        if (VCI_TransmitObj(BoardHdl, TxQueHdl, 0x206 , 2 , buf19) == VCI_OK)
                printf("ejecutar...\n");

        break;

        case '4':

                if (VCI_TransmitObj(BoardHdl, TxQueHdl, 0x306 , 8 , buf12) == VCI_OK)
                printf("primer punto\n");


                if (VCI_TransmitObj(BoardHdl, TxQueHdl, 0x306 , 8 , buf13) == VCI_OK)
                printf("segundo punto\n");


                if (VCI_TransmitObj(BoardHdl, TxQueHdl, 0x306 , 8 , buf14) == VCI_OK)
                printf("tercer punto\n");

                Sleep(5000);

                if (VCI_TransmitObj(BoardHdl, TxQueHdl, 0x606 , 8 , buf20) == VCI_OK)
                printf("posición actual\n");


                break;

        case '5':

        if (VCI_TransmitObj(BoardHdl, TxQueHdl, 0x306 , 8 , buf15) == VCI_OK)
                printf("4\n");

        if (VCI_TransmitObj(BoardHdl, TxQueHdl, 0x306 , 8 , buf22) == VCI_OK)
                printf("5\n");

        if (VCI_TransmitObj(BoardHdl, TxQueHdl, 0x306 , 8 , buf23) == VCI_OK)
                printf("6\n");

        break;

        case '5':

        if (VCI_TransmitObj(BoardHdl, TxQueHdl, 0x606 , 8 , buf17) == VCI_OK)
                printf("consumo\n");

        break;
         }

}

}

/*****************************************************************************
**
**   Function    : carga_puntos
**
**   Description : Función para cargar puntos al driver con id 6
**   Parameters  : -
**
**   Returnvalues: -
**
```

```
**************************************************************************/
void carga_puntos(void)
{
int tecla;
  static clock_t time = TX_CYCLE_TIME;
  static UINT8 buf0[2] = {0x01,0x06};
  static UINT8 buf1[2] = {0x06,0x00};
  static UINT8 buf2[2] = {0x07,0x00};
  static UINT8 buf3[2] = {0x0F,0x00}; //set to zero (enable)
  static UINT8 buf4[8] = {0x2F,0x01,0x16,0x00,0x00,0x00,0x00,0x00} ;
  static UINT8 buf5[8] = {0x23,0x01,0x16,0x01,0x20,0x01,0xC1,0x60} ;
  static UINT8 buf6[8] = {0x23,0x01,0x16,0x02,0x20,0x02,0xC1,0x60};
  static UINT8 buf7[8] = {0x2F,0x01,0x16,0x00,0x02,0x00,0x00,0x00};
  static UINT8 buf8[8] = {0x2F,0x60,0x60,0x00,0x07,0x00,0x00,0x00};
  static UINT8 buf9[8] = {0x2E,0xC0,0x60,0x00,0x00,0x00,0x00,0x00};
  static UINT8 buf10[8] = {0x2B,0x74,0x20,0x00,0x00,0x0C,0x00,0x00};
  static UINT8 buf11[8] = {0x23,0x79,0x20,0x00,0x00,0x00,0x00,0x00}; //initial position
  static UINT8 buf12[8] = {0x40,0x9c,0x00,0x00,0xd0,0x07,0x00,0x00}; //1st pt point
  static UINT8 buf13[8] = {0x00,0x00,0x00,0x00,0xd0,0x07,0x00,0x02}; //2nd pt point
  static UINT8 buf14[8] = {0x00,0x30,0x00,0x00,0xd0,0x07,0x00,0x04}; //3nd pt point
  static UINT8 buf15[8] = {0x00,0x40,0x00,0x00,0xd0,0x07,0x00,0x06}; //4th pt point
  static UINT8 buf16[8] = {0x23,0x64,0x20,0x00,0x85,0x00,0x30,0x02}; //configuración
lector registro
  static UINT8 buf17[8] = {0x40,0x66,0x20,0x00,0x00,0x00,0x00,0x00}; //lectura registro
  static UINT8 buf19[2] = {0x1F,0x00};
  static UINT8 buf20[8] = {0x40,0x64,0x60,0x00,0x00,0x00,0x00,0x00}; //Posicion actual
  static UINT8 buf21[8] = {0x40,0x40,0x60,0x00,0x00,0x00,0x00,0x00};


while(1){


        printf("1 - start remote node - ready to switch on - switch on\n");
     printf("2 - modo pt\n");
     printf("3 - ejecutar\n");
        printf("4 - puntos\n");



         tecla=getch();

        switch(tecla){

        case'1':

        if (VCI_TransmitObj(BoardHdl, TxQueHdl, 0x0   , 2 , buf0) == VCI_OK)
        printf("started remote node\n");


    if (VCI_TransmitObj(BoardHdl, TxQueHdl, 0x206 , 2 , buf1 ) == VCI_OK)
                printf("ready to switch on\n");


        if (VCI_TransmitObj(BoardHdl, TxQueHdl, 0x206 , 2 , buf2 ) == VCI_OK)
                 printf("switched on\n");
        Sleep(2000);

   if (VCI_TransmitObj(BoardHdl, TxQueHdl, 0x206 , 2 , buf3) == VCI_OK)
                printf("enabled\n");
```

```
        break;

        case'2':

  if (VCI_TransmitObj(BoardHdl, TxQueHdl, 0x606 , 8 , buf4) == VCI_OK)
                printf("set mode\n");

        if (VCI_TransmitObj(BoardHdl, TxQueHdl, 0x606 , 8 , buf5) == VCI_OK)
                printf("set vueltas\n");

        if (VCI_TransmitObj(BoardHdl, TxQueHdl, 0x606 , 8 , buf6) == VCI_OK)
                printf("set speed\n");

        if (VCI_TransmitObj(BoardHdl, TxQueHdl, 0x606 , 8 , buf7) == VCI_OK)
                printf("executing...\n");


        if (VCI_TransmitObj(BoardHdl, TxQueHdl, 0x606 , 8 , buf8) == VCI_OK)
                printf("reciving demand position value...\n");


        if (VCI_TransmitObj(BoardHdl, TxQueHdl, 0x606 , 8 , buf10) == VCI_OK)
                printf("consumo\n");

        if (VCI_TransmitObj(BoardHdl, TxQueHdl, 0x606 , 8 , buf16) == VCI_OK)
                printf("configurado lector registro\n");


        break;

        case '3':

        if (VCI_TransmitObj(BoardHdl, TxQueHdl, 0x606 , 8 , buf11) == VCI_OK)
                printf("posicion inicial...\n");

        if (VCI_TransmitObj(BoardHdl, TxQueHdl, 0x206 , 2 , buf19) == VCI_OK)
                printf("ejecutar...\n");

        if (VCI_TransmitObj(BoardHdl, TxQueHdl, 0x606 , 8 , buf21) == VCI_OK)
                printf("palabra de control...\n");

        break;

        case '4':

                if (VCI_TransmitObj(BoardHdl, TxQueHdl, 0x306 , 8 , buf12) == VCI_OK)
                printf("primer punto\n");

        Sleep(1000);


        if (VCI_TransmitObj(BoardHdl, TxQueHdl, 0x606 , 8 , buf20) == VCI_OK)
                printf("posición actual\n");


        if (VCI_TransmitObj(BoardHdl, TxQueHdl, 0x606 , 8 , buf17) == VCI_OK)
                printf("consumo\n");
```

```
        if (VCI_TransmitObj(BoardHdl, TxQueHdl, 0x306 , 8 , buf13) == VCI_OK)
                printf("segundo punto\n");
/*
        if (VCI_TransmitObj(BoardHdl, TxQueHdl, 0x306 , 8 , buf14) == VCI_OK)
                printf("tercer punto\n");

        if (VCI_TransmitObj(BoardHdl, TxQueHdl, 0x306 , 8 , buf15) == VCI_OK)
                printf("cuarto punto\n");

*/

        break;
         }

}

}

const char* Num2Function[] =
{
  "VCI_Init",
  "VCI_Searchboard",
  "VCI_Prepareboard",
  "VCI_Cancel_board",
  "VCI_Testboard",
  "VCI_ReadBoardInfo",
  "VCI_ReadBoardStatus",
  "VCI_Resetboard",
  "VCI_ReadCANInfo",
  "VCI_ReadCANStatus",
  "VCI_InitCAN",
  "VCI_SetAccMask",
  "VCI_ResetCAN",
  "VCI_StartCAN",
  "VCI_ResetTimeStamps",
  "VCI_ConfigQueue",
  "VCI_AssignRxQueObj",
  "VCI_ConfigBuffer",
  "VCI_ReconfigBuffer",
  "VCI_ConfigTimer",
  "VCI_ReadQueStatus",
  "VCI_ReadQueObj",
  "VCI_ReadBufStatus",
  "VCI_ReadBufData",
  "VCI_TransmitObj",
  "VCI_RequestObj",
  "VCI_UpdateBufObj",
  "VCI_CciReqData"
};


/*****************************************************************************
**
**   Function    : exception_handler
**
**   Description : called on exception within VCI
**   Parameters  : func_num: number identifying VCI function
**                 err_code: error code
**                 ext_err : extended error code
```

```
**              err_str : error description
**
**    Returnvalues: -
**
*****************************************************************************/
void VCI_CALLBACKATTR exception_handler (VCI_FUNC_NUM func_num,
                                INT32       err_code,
                                UINT16       ext_err,
                                char       *err_str )
{
  printf("Exception: %s (%i / %u) [%s]\n", Num2Function[func_num], err_code, ext_err,
err_str);
}

/*****************************************************************************
**
**    Function    : msg_handler
**
**    Description : called by VCI for status messages
**    Parameters  : -
**
**    Returnvalues: -
**
*****************************************************************************/
void VCI_CALLBACKATTR message_handler( char *msg_str )
{
  printf("Message: [%s]\n", msg_str);
}

/*****************************************************************************
**
**    Function    : receive_queuedata
**
**    Description : Receive - Callback Function
**    Parameters  : que_hdl: receive-queue handle
**              count  : number of can messages received
**              p_obj  : pointer to first can message
**                     (interrupt mode only)
**
**    Returnvalues: -
**
*****************************************************************************/
void VCI_CALLBACKATTR receive_queuedata( UINT16 que_hdl,
                                UINT16 count,
                                VCI_CAN_OBJ FAR * p_obj)
{
  static VCI_CAN_OBJ s_obj;
  int j,i;

  for (i = 0; i < count; i++)
  {
    memcpy(&s_obj,&(p_obj[i]),sizeof(VCI_CAN_OBJ));

    printf("\nTime: %10u Id: %3X   DLC: %1u    Data:",
s_obj.time_stamp,s_obj.id,s_obj.len);
    for(j=0;j<s_obj.len;j++)
    {
      printf(" %.2X",s_obj.a_data[j]);
    }
  }
```

```
}

/*****************************************************************************
**
**   Function    : receive_buffer_data
**
**   Description : function to read the configured Buffer CanObjects
**   Parameters  : -
**
**   Returnvalues: -
**
*****************************************************************************/
void receive_buffer_data()
{
  UINT8 buffer[8];
  UINT8 len;
  int i, j, ret;

  for(i = 0; i < ANZ_BUFFERS; i++)
  {
    if((ret = VCI_ReadBufData(BoardHdl, BufHdl[i], buffer, &len)) > 0)
    {
      printf("\n Count:%3u Id: %3X   DLC: %1u    Data:", ret, 0x586+i, len);
      for(j=0;j<len;j++)
      {
        printf(" %.2X\n",buffer[j]);
      }
    }
  }
}

/*****************************************************************************
**
**   Function    : end_app
**
**   Description : close the application
**   Parameters  : void:
**
**   Returnvalues: -
**
*****************************************************************************/
void end_app()
{
 // clrscr();
  VCI_CancelBoard(BoardHdl);
}
```

Anexo 2: Otras figuras y tablas

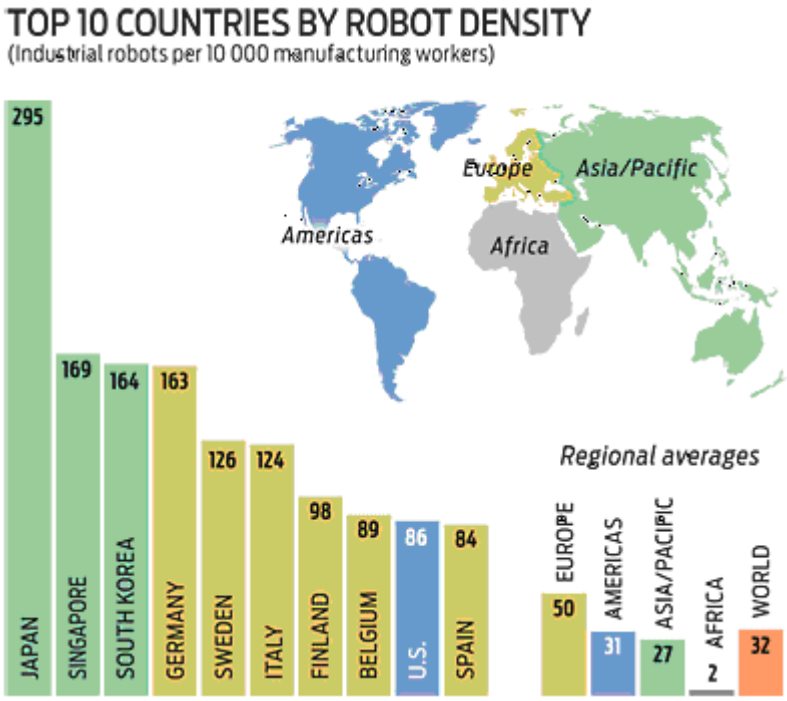


Figura 6.1: Densidad de robots por 10000 trabajadores

Velocidad Bus (Mbps)					
		Configuración A			Configuración B
tº computo	tº ejecucion(us)	6 driver/red (3 sms/ciclo)	12 drivers (3sms/ciclo)	24 encoders (2sms/consulta)	6 driver + 6 encoders (3+2) sms/ciclo
0	390	6,00	12	16	10,00
200	590	3,97	7,93	10,58	6,61
400	790	2,96	5,92	7,90	4,94
600	990	2,36	4,73	6,30	3,94
800	1190	1,97	3,93	5,24	3,28
1000	1390	1,68	3,37	4,49	2,81
1200	1590	1,47	2,94	3,92	2,45
1400	1790	1,31	2,61	3,49	2,18
1600	1990	1,18	2,35	3,14	1,96
1800	2190	1,07	2,14	2,85	1,78
2000	2390	0,98	1,96	2,61	1,63
2200	2590	0,90	1,81	2,41	1,51
2400	2790	0,84	1,68	2,24	1,40
2600	2990	0,78	1,57	2,09	1,30
2800	3190	0,73	1,47	1,96	1,22
3000	3390	0,69	1,38	1,84	1,15
3200	3590	0,65	1,30	1,74	1,09
3400	3790	0,62	1,23	1,65	1,03
3600	3990	0,59	1,17	1,56	0,98
3800	4190	0,56	1,12	1,49	0,93
4000	4390	0,53	1,07	1,42	0,89
4200	4590	0,51	1,02	1,36	0,85
4400	4790	0,49	0,98	1,30	0,81
4600	4990	0,47	0,94	1,25	0,78
4800	5190	0,45	0,90	1,20	0,75
5000	5390	0,43	0,87	1,16	0,72
5200	5590	0,42	0,84	1,12	0,70
5400	5790	0,40	0,81	1,08	0,67
5600	5990	0,39	0,78	1,04	0,65
5800	6190	0,38	0,76	1,01	0,63
6000	6390	0,37	0,73	0,98	0,61
6200	6590	0,36	0,71	0,95	0,59
6400	6790	0,34	0,69	0,92	0,57
6600	6990	0,33	0,67	0,89	0,56
6800	7190	0,33	0,65	0,87	0,54
7000	7390	0,32	0,63	0,84	0,53
7200	7590	0,31	0,62	0,82	0,51
7400	7790	0,30	0,60	0,80	0,50
7600	7990	0,29	0,59	0,78	0,49
7800	8190	0,29	0,57	0,76	0,48
8000	8390	0,28	0,56	0,74	0,46

Tabla 6.1: Velocidad Bus requerida en función tiempos de ejecución

Pruebas de consulta de intensidad:

Como se puede observar en las siguientes figuras, en las consultas que se hacen para ver el flujo de intensidad, el driver introduce un offset extraño que hace que el valor de intensidad recogido pueda ser erróneo. No hay que tener en cuenta los 4 bytes más significativos pues solo dan información del índice y subíndice del objeto al que se realiza la consulta:

Mensajes devueltos para un consumo de 1 A	**Mensajes devueltos para un consumo de 200 mA**
Data: 43 66 20 00 80 D5 E3 27 Data: 43 66 20 00 C0 E3 A3 19 Data: 43 66 20 00 00 29 91 D5 Data: 43 66 20 00 00 15 91 E9 Data: 43 66 20 00 80 D7 11 27 Data: 43 66 20 00 00 02 91 FC Data: 43 66 20 00 00 28 E0 D7 Data: 43 66 20 00 80 FC 60 03 Data: 43 66 20 00 40 D9 A0 26 Data: 43 66 20 00 00 DD 80 22 Data: 43 66 20 00 40 F5 83 07 Data: 43 66 20 00 80 D3 43 29 Data: 43 66 20 00 00 EC C3 10 Data: 43 66 20 00 40 1F 00 E0 Data: 43 66 20 00 C0 F3 80 0B Data: 43 66 20 00 00 DF 40 20 Data: 43 66 20 00 00 0D 40 F2 Data: 43 66 20 00 80 1D 1F E3 Data: 43 66 20 00 40 E0 5F 20 Data: 43 66 20 00 80 E3 1F 1D Data: 43 66 20 00 40 10 5F F0 Data: 43 66 20 00 C0 1B FF E4 Data: 43 66 20 00 00 F1 BF 0F Data: 43 66 20 00 C0 E5 FF 1A	Data: 43 66 20 00 C0 FB 52 02 Data: 43 66 20 00 40 FC D2 01 Data: 43 66 20 00 80 FF 92 FE Data: 43 66 20 00 80 FF 2F 01 Data: 43 66 20 00 40 EA 3E 17 Data: 43 66 20 00 C0 06 31 F8 Data: 43 66 20 00 C0 EC 31 12 Data: 43 66 20 00 C0 E7 31 17 Data: 43 66 20 00 C0 06 31 F8 Data: 43 66 20 00 80 16 71 E8 Data: 43 66 20 00 00 F6 E1 08 Data: 43 66 20 00 00 E7 E1 17 Data: 43 66 20 00 40 14 A1 EA Data: 43 66 20 00 80 0F 61 EF Data: 43 66 20 00 C0 F3 62 0A Data: 43 66 20 00 C0 ED 62 10 Data: 43 66 20 00 00 14 22 EA Data: 43 66 20 00 C0 FF 4F 00 Data: 43 66 20 00 80 E8 8F 17 Data: 43 66 20 00 C0 F2 4F 0D Data: 43 66 20 00 00 19 0F E7 Data: 43 66 20 00 40 FC 3E 05 Data: 43 66 20 00 40 00 E2 FD Data: 43 66 20 00 80 FD A2 00 Data: 43 66 20 00 00 FC 22 02

Tabla 6.2: Datos de los mensajes devueltos en consulta de intensidad

En la tabla siguiente se muestran los mensajes que devuelve el driver cuando se le hace una consulta del valor de intensidad y el motor está parado. Como se puede comprobar, hay ciertos bytes de datos que en este se ponen a 0 por lo que la consulta de la intensidad se ejecuta de forma correcta. Por ello habría que averiguar cómo se está proporcionando realmente la información.

Mensajes devueltos para un consumo de 0 A
Data: 43 66 20 00 40 00 C0 FF
Data: 43 66 20 00 80 FF 80 00
Data: 43 66 20 00 00 00 00 00
Data: 43 66 20 00 80 00 80 FF
Data: 43 66 20 00 00 00 00 00
Data: 43 66 20 00 80 FF 80 00
Data: 43 66 20 00 00 00 00 00
Data: 43 66 20 00 80 00 80 FF
Data: 43 66 20 00 C0 FF 40 00
Data: 43 66 20 00 80 FF 80 00
Data: 43 66 20 00 80 00 80 FF
Data: 43 66 20 00 00 00 00 00
Data: 43 66 20 00 80 FF 80 00
Data: 43 66 20 00 C0 FF 40 00
Data: 43 66 20 00 80 00 80 FF
Data: 43 66 20 00 80 00 80 FF
Data: 43 66 20 00 80 00 80 FF
Data: 43 66 20 00 80 00 80 FF
Data: 43 66 20 00 C0 FF 40 00
Data: 43 66 20 00 80 FF 80 00
Data: 43 66 20 00 80 00 80 FF
Data: 43 66 20 00 00 00 00 00
Data: 43 66 20 00 C0 FF 40 00
Data: 43 66 20 00 00 00 00 00
Data: 43 66 20 00 40 00 C0 FF

Tabla 6.3: Datos de los mensajes devueltos en consulta de intensidad con motor parado

Index	Sub-index	Description	Available for MotionChip III family?	Available for MotionChip II family?
1000h	00h	Device type	YES	YES
1001h	00h	Error register	YES	YES
1002h	00h	Manufacturer status register	YES	YES
1003h		Predefined error field	YES	YES
	00h	Number of errors in history	YES	YES
	01h	Standard error field (history 1)	YES	YES
	02h	Standard error field (history 2)	YES	YES
	03h	Standard error field (history 3)	YES	YES
	04h	Standard error field (history 4)	YES	YES
	05h	Standard error field (history 5)	YES	YES
1005h	00h	COB-ID of the SYNC message	YES	YES
1006h	00h	Communication cycle period	YES	YES
1008h	00h	Manufacturer device name	YES	NO
100Ah	00h	Manufacturer software version	YES	NO
100Ch	00h	Guard time	YES	YES
100Dh	00h	Lifetime factor	YES	YES
1013h	00h	High resolution time stamp	YES	YES
1014h	00h	COB-ID Emergency object	YES	YES
1017h	00h	Producer heartbeat time	YES	YES
1018h		Identity Object	YES	YES
	00h	Number of entries	YES	YES
	01h	Vendor ID	YES	YES
1200h		Server SDO parameter	YES	YES
	00h	Number of entries	YES	YES
	01h	COB-ID Client -> Server (rx)	YES	YES
	00h	Server SDO parameter	YES	YES
1400h		Receive PDO1 communication parameters	YES	YES
	00h	Number of entries	YES	YES
	01h	COB-ID RPDO1	YES	YES
	02h	Transmission type	YES	YES
1401h		Receive PDO2 communication parameters	YES	YES

	00h	Number of entries	YES	YES
	01h	COB-ID RPDO2	YES	YES
	02h	Transmission type	YES	YES
1402h		Receive PDO3 communication parameters	YES	YES
	00h	Number of entries	YES	YES
	01h	COB-ID RPDO3	YES	YES
	02h	Transmission type	YES	YES
1403h		Receive PDO4 communication parameters	YES	YES
	00h	Number of entries	YES	YES
	01h	COB-ID RPDO4	YES	YES
	02h	Transmission type	YES	YES
1600h		RPDO1 mapping parameters	YES	YES
	00h	Number of entries	YES	YES
	01h	1st mapped object – 6040h – control word	YES	YES
1601h		RPDO2 mapping parameters	YES	YES
	00h	Number of entries	YES	YES
	01h	1st mapped object – 6040h – control word	YES	YES
	02h	2nd mapped object – 6060h – modes of operation	YES	YES
1602h		RPDO3 mapping parameters	YES	YES
	00h	Number of entries	YES	YES
	01h	1st mapped object – 6040h – control word	YES	YES
	02h	2nd mapped object – 607Ah – target position	YES	YES
1603h		RPDO4 mapping parameters	YES	YES
	00h	Number of entries	YES	YES
	01h	1st mapped object – 6040h – control word	YES	YES
	02h	2nd mapped object – 60FFh – target velocity	YES	YES
1800h		TPDO1 communication parameters	YES	YES
	00h	Number of entries	YES	YES
	01h	COB-ID TPDO1	YES	YES
	02h	Transmission type	YES	YES
	03h	Reserved	YES	YES
	04h	Reserved	YES	YES
	05h	Event timer	YES	YES
1801h		TPDO2 communication parameters	YES	YES
	00h	Number of entries	YES	YES

	01h	COB-ID TPDO2	YES	YES
	02h	Transmission type	YES	YES
	03h	Reserved	YES	YES
	04h	Reserved	YES	YES
	05h	Event timer	YES	YES
1802h		TPDO3 communication parameters	YES	YES
	00h	Number of entries	YES	YES
	01h	COB-ID TPDO3	YES	YES
	02h	Transmission type	YES	YES
	03h	Reserved	YES	YES
	04h	Reserved	YES	YES
	05h	Event timer	YES	YES
1803h		TPDO4 communication parameters	YES	YES
	00h	Number of entries	YES	YES
	01h	COB-ID TPDO4	YES	YES
	02h	Transmission type	YES	YES
	03h	Reserved	YES	YES
	04h	Reserved	YES	YES
	05h	Event timer	YES	YES
1A00h		TPDO1 mapping parameters	YES	YES
	00h	Number of entries	YES	YES
	01h	1st mapped object – 6041h – status word	YES	YES
1A01h		TPDO2 mapping parameters	YES	YES
	00h	Number of entries	YES	YES
	01h	1st mapped object – 6041h – status word	YES	YES
	02h	2nd mapped object – 6061h – modes of operation display	YES	YES
1A02h		TPDO3 mapping parameters	YES	YES
	00h	Number of entries	YES	YES
	01h	1st mapped object – 6041h – status word	YES	YES
	02h	2nd mapped object – 6064h – position actual value	YES	YES
1A03h		TPDO4 mapping parameters	YES	YES
	00h	Number of entries	YES	YES
	01h	1st mapped object – 606Bh – velocity demand value	YES	YES

	02h	2nd mapped object – 606Ch – velocity actual value	YES	YES
2000h	00h	Motion Error Register	YES	YES
2001h	00h	Motion Error Register mask	YES	YES
2002h	00h	Drive status mask	YES	YES
2004h	00h	COB-ID High resolution time stamp	YES	YES
2005h	00h	Max slippage time out	YES	NO
2006h	00h	Call TML function	YES	YES
2010h	00h	Master settings	YES	NO
2012h	00h	Master resolution	YES	NO
2013h		EGEAR multiplication factor	YES	NO
	00h	Number of entries	YES	NO
	01h	EGEAR ratio numerator (slave)	YES	NO
	02h	EGEAR ratio denominator (master)	YES	NO
2017h	00h	Master actual position	YES	NO
2018h	00h	Master actual speed	YES	NO
2019h	00h	CAM table load address	YES	NO
201Ah	00h	CAM table run address	YES	NO
201Bh	00h	CAM offset	YES	NO
201Ch	00h	External on-line reference	YES	NO
201Dh	00h	External reference type	YES	NO
2022h	00h	Control effort	YES	YES
2023h	00h	Jerk time	YES	YES
2025h	00h	Stepper current in open loop operation	YES	YES
2026h	00h	Stand-by current for stepper in open loop operation	YES	YES
2027h	00h	Timeout for stepper stand-by current	YES	YES
2040h	00h	Digital inputs status	NO	YES
2042h	00h	Digital inputs mask	YES	NO
2043h	00h	Digital outputs command	NO	YES
2045h	00h	Digital outputs status	YES	NO
2046h	00h	Analogue input: Reference	YES	NO
2047h	00h	Analogue input: Feedback	YES	NO
2050h	00h	Over current protection level	YES	YES
2051h	00h	Over current time out	YES	YES
2052h	00h	Motor nominal current	YES	YES
2053h	00h	I2t protection integrator limit	YES	YES

2054h	00h	I2t protection scaling factor	YES	YES
2055h	00h	DC-link voltage	YES	NO
2058h	00h	Drive temperature	YES	NO
2060h	00h	Software version of the TML application	YES	NO
2064h	00h	Read/Write configuration register	YES	YES
2065h	00h	Write data at address set in object 2064h (16/32 bits)	YES	YES
2066h	00h	Read data from address set in object 2064h (16/32 bits)	YES	YES
2067h	00h	Write data at specified address	YES	YES
2069h	00h	Checksum configuration register	YES	YES
206Ah	00h	Checksum read register	YES	YES
206Bh	00h	CAM input scaling factor	YES	NO
206Ch	00h	CAM output scaling factor	YES	NO
206Fh	00h	Time notation index	YES	NO
2070h	00h	Time dimension index	YES	NO
2071h		Time factor	YES	NO
	00h	Number of entries	YES	NO
	01h	Numerator	YES	NO
	02h	Divisor	YES	NO
2072h	00h	Interpolated position mode status	YES	YES
2073h	00h	Interpolated position buffer length	YES	YES
2074h	00h	Interpolated position buffer configuration	YES	YES
2075h		Position triggers	YES	NO
	00h	Number of entries	YES	NO
	01h	Position trigger 1	YES	NO
	02h	Position trigger 2	YES	NO
	03h	Position trigger 3	YES	NO
	04h	Position trigger 4	YES	NO
2076h	00h	Save current configuration	YES	YES
2077h	00h	Execute TML program	YES	YES
2078h	00h	Execute auto-tuning for Linear Halls config	YES	NO
2079h	00h	Interpolated position initial position	YES	YES
6007h	00h	Abort connection option code	YES	NO
6040h	00h	Control word	YES	YES
6041h	00h	Status word	YES	YES
605Eh	00h	Fault reaction option code	YES	NO

6060h	00h	Modes of operation	YES	YES
6061h	00h	Modes of operation display	YES	YES
6062h	00h	Position demand value	YES	YES
6063h	00h	Position actual value*	YES	NO
6064h	00h	Position actual value	YES	YES
6065h	00h	Following error window	YES	YES
6066h	00h	Following error time out	YES	YES
6067h	00h	Position window	YES	YES
6068h	00h	Position window time	YES	YES
6069h	00h	Velocity sensor actual value	YES	YES
606Bh	00h	Velocity demand value	YES	YES
606Ch	00h	Velocity actual value	YES	YES
606Fh	00h	Velocity threshold	YES	NO
607Ah	00h	Target position	YES	YES
607Ch	00h	Home offset	YES	YES
6081h	00h	Profile velocity	YES	YES
6083h	00h	Profile acceleration	YES	YES
6085h	00h	Quick stop deceleration	YES	YES
6086h	00h	Motion profile type	YES	YES
6089h	00h	Position notation index	YES	NO
608Ah	00h	Position dimension index	YES	NO
608Bh	00h	Velocity notation index	YES	NO
608Ch	00h	Velocity dimension index	YES	NO
608Dh	00h	Acceleration notation index	YES	NO
608Eh	00h	Acceleration dimension index	YES	NO
6093h		Position factor	YES	NO
	00h	Number of entries	YES	NO
	01h	Numerator	YES	NO
	02h	Divisor	YES	NO
6095h		Velocity encoder factor	YES	NO
	00h	Number of entries	YES	NO
	01h	Numerator	YES	NO
	02h	Divisor	YES	NO
6097h		Acceleration factor	YES	NO
	00h	Number of entries	YES	NO
	01h	Numerator	YES	NO
	02h	Divisor	YES	NO

Tabla 6.4: Diccionario de objeto del driver ISCM8005

Anexo 3: Hojas características

Motor Faulhaber

DC-Micromotors

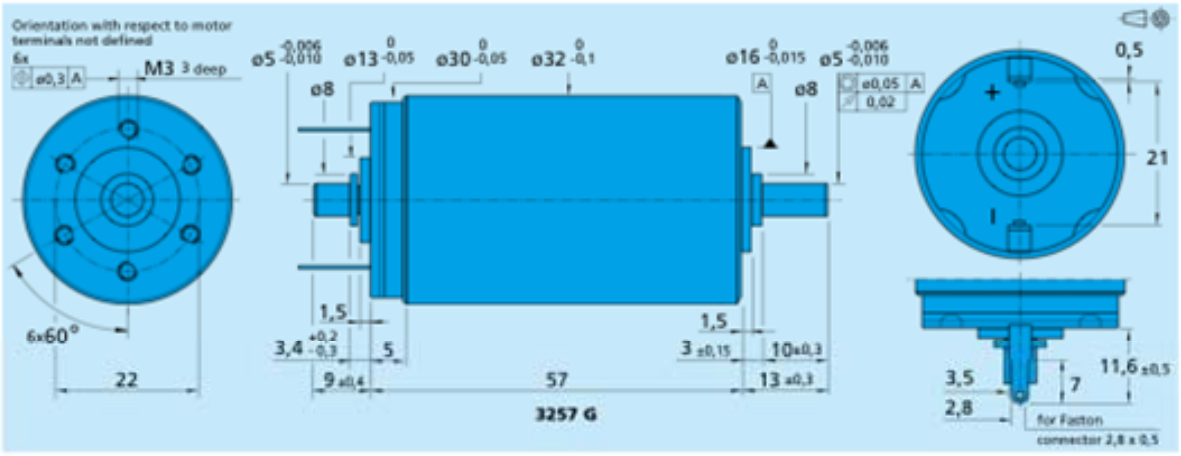
Graphite Commutation

70 mNm

For combination with (overview on page 14-15)
Gearheads:
32/3, 38/1, 38/2
Encoders:
IE2 – 16 ... 512, 5500, 5540

Series 3257 ... CR

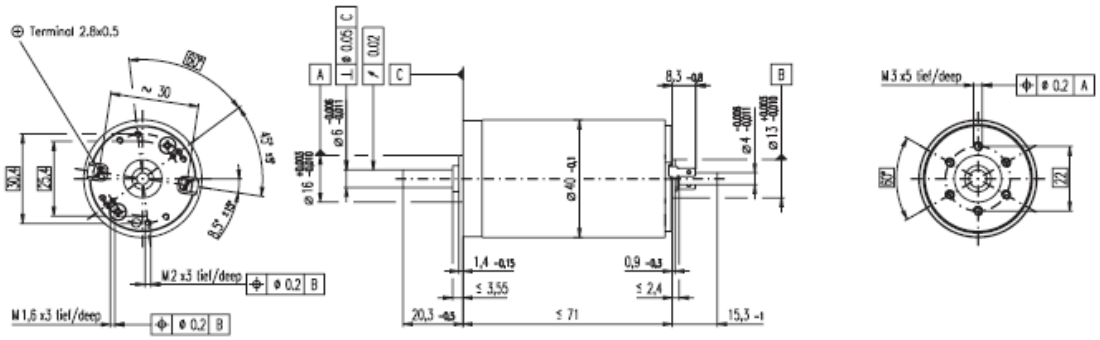
		3257 G		012 CR	024 CR	048 CR	
1	Nominal voltage	U_N		12	24	48	Volt
2	Terminal resistance	R		0,41	1,63	6,56	Ω
3	Output power	$P_{2\,max.}$		79,2	83,2	84,5	W
4	Efficiency	$\eta_{max.}$		83	83	83	%
5	No-load speed	n_0		5 700	5 900	5 900	rpm
6	No-load current (with shaft ø 5,0 mm)	I_0		0,258	0,129	0,064	A
7	Stall torque	M_H		531	539	547	mNm
8	Friction torque	M_R		4,9	4,9	4,9	mNm
9	Speed constant	k_n		500	253	125	rpm/V
10	Back-EMF constant	k_E		2,00	3,95	7,98	mV/rpm
11	Torque constant	k_M		19,1	37,7	76,2	mNm/A
12	Current constant	k_I		0,052	0,027	0,013	A/mNm
13	Slope of n-M curve	Δn/ΔM		10,7	10,9	10,8	rpm/mNm
14	Rotor inductance	L		70	270	1 100	µH
15	Mechanical time constant	τ_m		4,7	4,7	4,7	ms
16	Rotor inertia	J		42	41	42	gcm²
17	Angular acceleration	$\alpha_{max.}$		130	130	130	10^3rad/s²
18	Thermal resistance	$R_{th\,1}$ / $R_{th\,2}$	2 / 8				K/W
19	Thermal time constant	τ_{w1} / τ_{w2}	17 / 810				s
20	Operating temperature range:						
	– motor		– 30 ... + 125				°C
	– rotor, max. permissible		+ 155				°C
21	Shaft bearings		ball bearings, preloaded				
22	Shaft load max.:						
	– with shaft diameter		5,0				mm
	– radial at 3 000 rpm (3 mm from bearing)		50				N
	– axial at 3 000 rpm		5				N
	– axial at standstill		50				N
23	Shaft play:						
	– radial	≤	0,015				mm
	– axial	=	0				mm
24	Housing material		steel, black coated				
25	Weight		242				g
26	Direction of rotation		clockwise, viewed from the front face				
Recommended values - mathematically independent of each other							
27	Speed up to	$n_{max.}$		5 000	5 000	5 000	rpm
28	Torque up to	$M_{max.}$		70	70	70	mNm
29	Current up to (thermal limits)	$I_{max.}$		4,60	2,30	1,15	A



Motor Maxon

maxon DC motor

RE 40 ∅40 mm, Graphite Brushes, 150 Watt



M 1:2

Stock program
Standard program
Special program (on request)

			Order Number										
			148866	148867	148877	218008	218009	218010	218011	218012	218013	218014	218015
Motor Data													
	Values at nominal voltage												
1	Nominal voltage	V	12.0	24.0	48.0	48.0	48.0	48.0	48.0	48.0	48.0	48.0	48.0
2	No load speed	rpm	6920	7580	7580	6420	5560	3330	2690	2130	1710	1420	987
3	No load current	mA	241	137	68.6	53.7	43.7	21.9	16.7	12.5	9.67	7.77	5.16
4	Nominal speed	rpm	6370	6930	7000	5810	4920	2700	2050	1500	1080	774	339
5	Nominal torque (max. continuous torque)	mNm	94.9	170	184	183	177	187	187	189	189	188	188
6	Nominal current (max. continuous current)	A	6.00	5.77	3.12	2.62	2.20	1.38	1.12	0.898	0.721	0.593	0.413
7	Stall torque	mNm	1680	2280	2500	1990	1590	995	796	641	512	415	289
8	Starting current	A	102	75.7	41.4	28.0	19.2	7.26	4.68	3.00	1.92	1.29	0.627
9	Max. efficiency	%	88	91	92	91	91	89	88	87	86	85	83
	Characteristics												
10	Terminal resistance	Ω	0.117	0.317	1.16	1.72	2.50	6.61	10.2	16.0	24.9	37.1	76.6
11	Terminal inductance	mH	0.0245	0.0823	0.329	0.460	0.612	1.70	2.62	4.14	6.40	9.31	19.2
12	Torque constant	mNm / A	16.4	30.2	60.3	71.3	82.2	137	170	214	266	321	461
13	Speed constant	rpm / V	581	317	158	134	116	69.7	56.2	44.7	35.9	29.8	20.7
14	Speed / torque gradient	rpm / mNm	4.15	3.33	3.04	3.23	3.53	3.36	3.39	3.35	3.37	3.44	3.45
15	Mechanical time constant	ms	6.03	4.81	4.39	4.36	4.35	4.31	4.31	4.31	4.31	4.32	4.33
16	Rotor inertia	gcm²	139	138	138	129	118	123	121	123	122	120	120

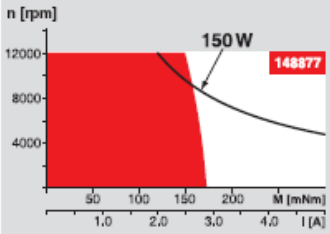
Specifications

	Thermal data	
17	Thermal resistance housing-ambient	4.65 K / W
18	Thermal resistance winding-housing	1.93 K / W
19	Thermal time constant winding	41.6 s
20	Thermal time constant motor	1120 s
21	Ambient temperature	-30 ... +100°C
22	Max. permissible winding temperature	+155°C
	Mechanical data (ball bearings)	
23	Max. permissible speed	12000 rpm
24	Axial play	0.05 - 0.15 mm
25	Radial play	0.025 mm
26	Max. axial load (dynamic)	5.6 N
27	Max. force for press fits (static)	110 N
	(static, shaft supported)	1200 N
28	Max. radial loading, 5 mm from flange	28 N
	Other specifications	
29	Number of pole pairs	1
30	Number of commutator segments	13
31	Weight of motor	480 g

Values listed in the table are nominal.
Explanation of the figures on page 49.

Option
Preloaded ball bearings

Operating Range



Comments

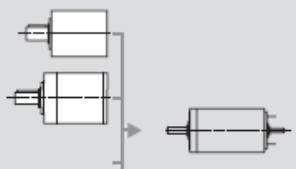
Continuous operation
In observation of above listed thermal resistance (lines 17 and 18) the maximum permissible winding temperature will be reached during continuous operation at 25°C ambient.
= Thermal limit.

Short term operation
The motor may be briefly overloaded (recurring).

Assigned power rating

maxon Modular System

Overview on page 16 - 21



Planetary Gearhead
∅42 mm
3 - 15 Nm
Page 244

Planetary Gearhead
∅52 mm
4 - 30 Nm
Page 247

Encoder MR
256 - 1024 CPT,
3 channels
Page 259

Encoder HED_ 5540
500 CPT,
3 channels
Page 262 / 264

Brake AB 28
∅45 mm
24 VDC, 0.4 Nm
Page 308

Industrial Version
Encoder HEDL 9140
Page 267
Brake AB 28
Page 309

Recommended Electronics:

ADS 50/5	Page 276
ADS 50/10	277
ADS_E 50/5	277
ADS_E 50/10	277
EPOS 24/5	294
EPOS2 50/5	295
EPOS 70/10	295
EPOS P 24/5	297
Notes	18

May 2009 edition / subject to change

Encoder Faulhaber

FAULHABER

Encoders

Optical Encoders with Line Driver

Features:
500 Lines per revolution
3 Channels + complementary outputs
Digital output
Line driver

Series 5540

		HEDL 5540	
Lines per revolution	N	500	
Signal output, square wave		2+1 index and complementary outputs	channels
Supply voltage	V_{CC}	4,5 ... 5,5	V DC
Current consumption, typical (V_{CC} = 5 V DC)	I_{CC}	57	mA
Pulse width	P	180 ± 35	°e
Index pulse width	P_O	90 ± 35	°e
Phase shift, channel A to B	Φ	90 ± 15	°e
Logic state width	S	90 ± 35	°e
Cycle	C	360 ± 5,5	°e
Signal rise/fall time, typical	tr/tf	0,25 / 0,25	µs
Frequency range [1]	f	up to 100	kHz
Inertia of code disc	J	0,6	gcm²
Operating temperature range		0 ... + 70	°C

[1] Velocity (rpm) = f (Hz) x 60/N

Ordering information

Encoder type	number of channels	lines per revolution	For combination with:
HEDL 5540 A	2+1	500	DC-Micromotors and DC-Motor-Tachos Series
			2230, 2233, 2251
			2342
			2642, 2657
			3242, 3257, 3557, 3863
			brushless DC-Servomotors Series
			2036, 2057, 2444, 3056, 3564

The housing dimensions of the HEDL encoder are the same as the HEDS/HEDM encoders, but there is a ribbon cable instead of plain connector pins.

Suggested Line Receivers: AM26LS32, SN75175, MC3486

Features

These incremental shaft encoders in combination with the DC-Micromotors and Brushless DC-Servomotors are designed for the indication and control of both shaft velocity and direction of rotation as well as for positioning.

A LED source and lens system transmits collimated light through a low inertia metal disc to give two channels with 90° phase shift.

The index pulse is synchronized with the channel $\overline{B}$.
Each encoder channel provides complementary output signals.

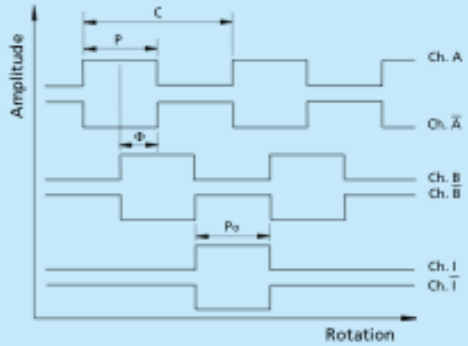
The single 5 volt supply and the digital output signals are interfaced with a connector.

The line driver offers enhanced performance when the encoder is used in noisy environments, or when it is required to drive long distances.

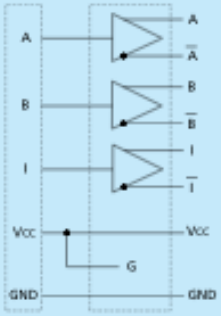
Motor with ball bearings are recommended for continuous operation at low and high speeds and for elevated radial shaft load.

Details for the motors and suitable reduction gearheads are on separate catalogue pages.

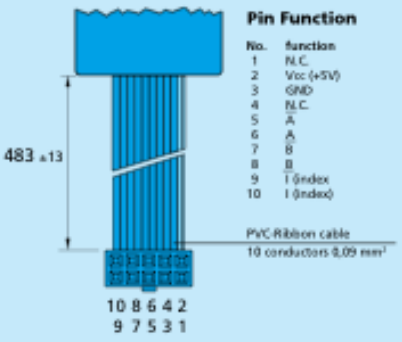
Output signals / Circuit diagram / Connector information



Output signals HEDL 5540
with clockwise rotation as seen from the shaft end



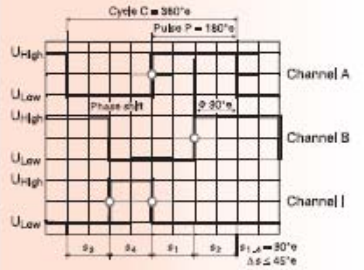
Connection diagram



Connector
DIN-41651
grid 2,54 mm

Encoder Maxon

Encoder MR, Type L, 256 - 1024 CPT, 3 Channels, with Line Driver



Stock program
Standard program
Special program (on request)

Order Number				
225783	226452	225785	226456	225787
Type				
Counts per turn	256	500	512	1000	1024
Number of channels	3	3	3	3	3
Max. operating frequency (kHz)	80	200	160	200	320

Combination

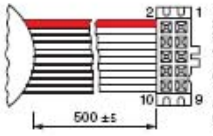
+ Motor	Page	+ Gearhead	Page	+ Brake	Page	Overall length [mm] / ● see: + Gearhead				
RE 30, 60 W	81					79.4	79.4	79.4	79.4	79.4
RE 30, 60 W	81	GP 32, 0.75 - 6.0 Nm	239/240			●	●	●	●	●
RE 35, 90 W	82					82.3	82.3	82.3	82.3	82.3
RE 35, 90 W	82	GP 32, 0.75 - 6.0 Nm	239/240			●	●	●	●	●
RE 35, 90 W	82	GP 32, 8 Nm	242			●	●	●	●	●
RE 35, 90 W	82	GP 42, 3 - 15 Nm	244			●	●	●	●	●
RE 36, 70 W	83					82.6	82.6	82.6	82.6	82.6
RE 36, 70 W	83	GP 32, 0.4 - 2.0 Nm	237			●	●	●	●	●
RE 36, 70 W	83	GP 32, 0.75 - 6.0 Nm	239/240			●	●	●	●	●
RE 36, 70 W	83	GP 42, 3 - 15 Nm	244			●	●	●	●	●
RE 40, 150 W	84					82.4	82.4	82.4	82.4	82.4
RE 40, 150 W	84	GP 42, 3 - 15 Nm	244			●	●	●	●	●
RE 40, 150 W	84	GP 52, 4 - 30 Nm	247			●	●	●	●	●
A-max 32	124/126					72.7	72.7	72.7	72.7	72.7
A-max 32	124/126	GP 32, 0.75 - 6.0 Nm	239/241			●	●	●	●	●
A-max 32	124/126	GS 38, 0.1 - 0.6 Nm	243			●	●	●	●	●
EC-max 40, 70 W	180						73.9		73.9	
EC-max 40, 70 W	180	GP 42, 3 - 15 Nm	245			●	●	●	●	●
EC-max 40, 120 W	181						103.9		103.9	
EC-max 40, 120 W	181	GP 52, 4 - 30 Nm	248				●		●	
EC-I 40, 50 W	199					42.0	42.0	42.0	42.0	42.0
EC-I 40, 50 W	199	GP 32, 1 - 6 Nm	241			●	●	●	●	●
EC-I 40, 70 W	200					52.0	52.0	52.0	52.0	52.0
EC-I 40, 70 W	200	GP 32, 1 - 6 Nm	241			●	●	●	●	●

Technical Data

Supply voltage	5 V ± 5 %
Output signal	TTL compatible
Index pulse width (nominal)	90°e ± 45°e
Operating temperature range	-25 ... +85°C
Moment of inertia of code wheel	≤ 1.7 gcm²
Output current per channel	max. 5 mA

The index signal I is synchronised with channel A or B.

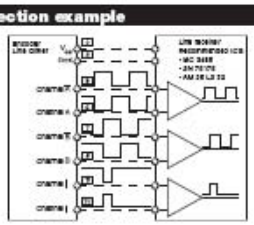
Pin Allocation



1	N.C.
2	V_{CC}
3	GND
4	N.C.
5	Channel $\overline{A}$
6	Channel A
7	Channel $\overline{B}$
8	Channel B
9	Channel $\overline{I}$ (Index)
10	Channel I (Index)

DIN Connector 41651
flat band cable AWG 28

Connection example



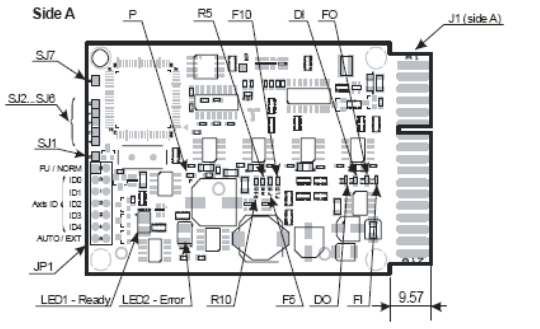
May 2008 edition / subject to change

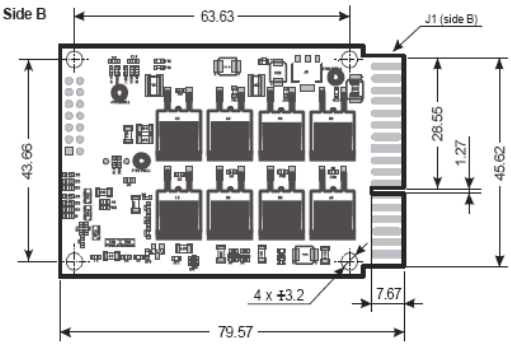
maxon tacho 259

Drver ISCM8005 de Technosoft

TECHNOSOFT

ISCM8005 DATASHEET
P/N: P047.001.E301





All dimensions in mm.

Motor – sensor configurations

Sensor \ Motor	PMSM	BLDC	DC BRUSH	STEP (2-ph.)	STEP (3-ph.)
Encoder	●		●	●	
Encoder + Hall	●	●			
Tacho			●		
Linear Halls	●				
Open-loop (no sensor)				●	●

Mating connector type

Reference	Producer	P/N	Description
J1	E-TEC	ECC-034-S100-01	Type: Wire crimp connector housing Description: Card Edge "Twin Leaf" connector, 0.1" pitch, 2 x 17 contacts, for wire crimp pins, with mounting ears. Requires also: 1 x intercontact keying plug (1-530030-2) Type: PCB connector

Features

- Continuous drive power – step motor: 200 W (48V; 5A)
- Motor supply: 12-80V_{DC}; 5A; 16A PEAK
- Logic supply: 12-48V_{DC}; 120 mA;
- Single-ended, differential and/or open-collector encoder interface
- Single-ended, open collector Hall sensor interface
- Differential pulse & direction
- Linear Hall sensor interface
- 7 Input-output lines:
  - 3 digital input-output lines
  - 2 digital input-output lines shared with 2 analog inputs (0 … 3.3 V)
- RESET input
- Emergency shutdown (ENABLE) input
- RS-232 serial interface, up to 115kbps communication speed
- CAN-Bus 2.0B interface up to 1Mbit/s
- Hardware Axis ID selection
- 1.5K × 16 internal SRAM memory
- 8K × 16 E^2ROM to store TML programs and data
- Operating ambient temperature: 0-40°C

Connectors description

	Pin	Name	Type	Description
J1 (side A)	A1	+Vmot	I	Positive terminal of the motor supply: 12 to 80V_{DC}
	A2	A / A+	O	• Phase A for brushless motors • Phase A+ for 2-phases step motors • Phase U for 3-phases step motors • Motor+ for DC brush motors
	A3	B / A-	O	• Phase B for brushless motors • Phase A- for 2-phases step motors • Phase V for 3-phases step motors • Motor- for DC brush motors
	A4	C / B+	O	• Phase C for brushless motors • Phase B+ for 2-phases step motors • Phase W for 3-phases step motors
	A5	BRAKE/B -	O	• Brake output (for external brake resistor) • Phase B- for 2-phases step motors
	A6	+5 V	O	5V logic supply (internally generated)
	A7	ENCA+	I	• Single-ended encoder A signal • Differential encoder positive A input
	A8	ENCB+	I	• Single-ended encoder B signal • Differential encoder positive B input
	A9	ENCZ / CAPI+	I	• Single-ended encoder Z signal • Differential encoder positive Z input
	A10	H1	I	• Hall 1 signal for digital Hall sensor
	A11	IO#38 / PULSE	I	• 5V compatible digital input / 3.3V compatible digital output • PULSE input in Pulse & Direction mode
	A12	IN#2 / LSP	I	• 5V compatible input • Positive limit switch
	A13	ENABLE	I	5V compatible. Connect to +5 V to disable PWM outputs
	A14	IO#13 / FDBK	I/O I	• 5V compatible digital input / 3.3V compatible digital output • Unipolar 0V…+5 V analog input. May be used as analog position or tacho speed feedback
	A15	GND	-	Ground
	A16	CAN_H	I/O	Can-Bus positive line (positive during dominant bit)
	A17	TX232	O	RS-232 Data Transmission

TECHNOSOFT

ISCM8005 DATASHEET
P/N: P047.001.E301

J1 (side B)	B1	+Vmot	I	Positive terminal of the motor supply: 12 to 48 V_{DC} for ISCM4805 12 to 80 V_{DC} for ISCM8005
	B2	A / A+	O	• Phase A for brushless motors • Phase A+ for 2-phases step motors • Phase U for 3-phases step motors • Motor+ for DC brush motors
	B3	B / A-	O	• Phase B for brushless motors • Phase A- for 2-phases step motors • Phase V for 3-phases step motors • Motor- for DC brush motors
	B4	C / B+	O	• Phase C for brushless motors • Phase B+ for 2-phases step motors • Phase W for 3-phases step motors
	B5	BRAKE / B-	O	• Brake output (for external brake resistor) for brushless motors • Phase B- for 2-phases step motors
	B6	+Vlog	I	Positive terminal of the logic supply: +12 to +48 V_{DC}
	B7	ENCA- /LH1	I	• Single-ended encoder A signal • Differential encoder negative A signal • Linear Hall 1 signal
	B8	ENCB- /LH2	I	• Single-ended encoder B signal • Differential encoder negative B signal • Linear Hall 2 signal
	B9	ENCZ- /LH3	I	• Single-ended encoder Z signal • Differential encoder negative Z signal • Linear Hall 3 signal
	B10	H2	I	• Hall 2 signal for digital Hall sensor
	B11	H3	I	• Hall 3 signal for digital Hall sensor
	B12	IN#24 / LSN	I	• 5V compatible input • Negative limit switch
	B13	RESET	I	• RESET signal – connect to +5 V to reset the board
	B14	IO#14 / REF / DIR	I/O I	• 5V compatible digital input / 3.3V compatible digital output • Unipolar 0 V…+5 V analog input. May be used as analog position, speed or torque reference. • Can be used as DIRECTION input in Pulse & Direction motion mode
	B15	GND	-	Ground
	B16	CAN_L	I/O	CAN-Bus negative line (negative during dominant bit)
	B17	RX232	I	RS-232 Data Reception

TECHNOSOFT

ISCM8005 DATASHEET
P/N: P047.001.E301

Electrical characteristics

All parameters are measured under the following conditions (unless otherwise noted):

- T_{amb} = 25°C, logic supply (+Vlog) = 24V_{DC}, motor supply (+Vmot) = 80V_{DC}
- Supplies start-up / shutdown sequence: *-any-* ;
- Load current 4A_{RMS}
- External DC-bus capacitor located 10 cm from J1.

		Min.	Typ.	Max.	Units
Logic Supply Input	Measured between +5V and GND.				
Supply voltage	Nominal values	12	24	48	V_{DC}
	Absolute maximum values, continuous †	-0.5		50	V_{DC}
Supply current	Normal operating		100	250	mA
Motor Supply Input	Measured between +V_{MOT} and GND.				
Supply voltage	Operating voltages, including ripple & braking-induced over-voltage	12		80	V
	Absolute maximum values, continuous †	0		95	V
	Absolute maximum values, surge (duration ≤ 10mS) †	-0.5		105	V
Supply current	Idle			1	mA
	Operating	-16.5	±5	+16.5	A
DC-bus capacitor value	DC-bus capacitor connected between +V_{MOT} and GND	100			µF
DC-bus capacitor location	Wire length from DC-bus capacitor to J1 pins	0	10	20	cm
Motor Outputs	All voltages referenced to GND.				
Motor output current	Continuous operation	-5		+5	A_{RMS}
Motor output current, peak	Thermal limited to <= 0.5 s	-16.5		+16.5	A
On-state voltage drop	Output current = ±5A	-800	±150	+250	V
Off-state leakage current		-1	±0.1	+1	mA
Motor inductance	F_{PWM} = 20kHz, +V_{MOT} = 12V	50			µH
	F_{PWM} = 20kHz, +V_{MOT} = 80V	400			µH

5V Digital Inputs		All voltages referenced to GND.				
Input voltage		Logic "LOW"	-	-	0.8	V
		Logic "HIGH"	2	-	5	V
		Absolute maximum, surge (duration ≤ 1S) †	-		+5.6	V
Input current	IO#35, IO#36	Logic "HIGH"; Internal pull-up to +5V	0	0	0	µA
		Logic "LOW"	-	-	20	µA
	PULS+, DIR+, ENCA, ENCB, RESET	Logic "HIGH"; Internal pull-up to +5V	0	0	0	µA
		Logic "LOW"	-	-	1000	µA
	PULS-, DIR-	Logic "HIGH"			1500	µA
		Logic "LOW"			700	µA
Input frequency			0		5	MHz
Minimum pulse width			150			nS
ESD Protection		Human Body Model (100 pF, 1.5KΩ)			±2	KV

Analog Inputs	All voltages referred to GND.				
Resolution			10		bits
Differential linearity	Guaranteed 10-bits no-missing-codes			0.09	% FS[1]
Offset error	Common-mode voltage = 0…10V		±0.1	±0.3	% FS[1]
Gain error	Common-mode voltage = 0…10V		±0.5	±1	% FS[1]
Bandwidth (-3dB)			1		KHz
Input voltage Operating range		0		3.3	V
Input impedance			50		KΩ
ESD Protection	Human Body Model (100 pF, 1.5 KΩ)			±2	KV

RS-232	All voltages referred to GND.				
Standards compliance		TIA/EIA-232-C			
Bit rate	Depending on software settings	9600		115200	Baud
ESD Protection	Human Body Model (100 pF, 1.5 kΩ)			±15	KV
Input voltage	RX232 input	-25	-	+25	V
Output short-circuit withstand	TX232 output to GND	Guaranteed			

CAN-Bus	All voltages referred to GND.				
Standards compliance		CAN-Bus 2.0B error active; ISO 11898-2			
Transmission line impedance	Recommended; Measured at 1MHz	90	120	150	Ω
Bit rate	Depending on SW settings	125		1000	Kbps
Number of network nodes	Depending on SW settings			64	-
ESD Protection	Human Body Model			±15	KV
Others					
Operating temperature		0		+40	°C
Weight			50		g
Storage temperature	Not powered	-40		85	°C
Humidity	Non-condensing	0		90	%RH

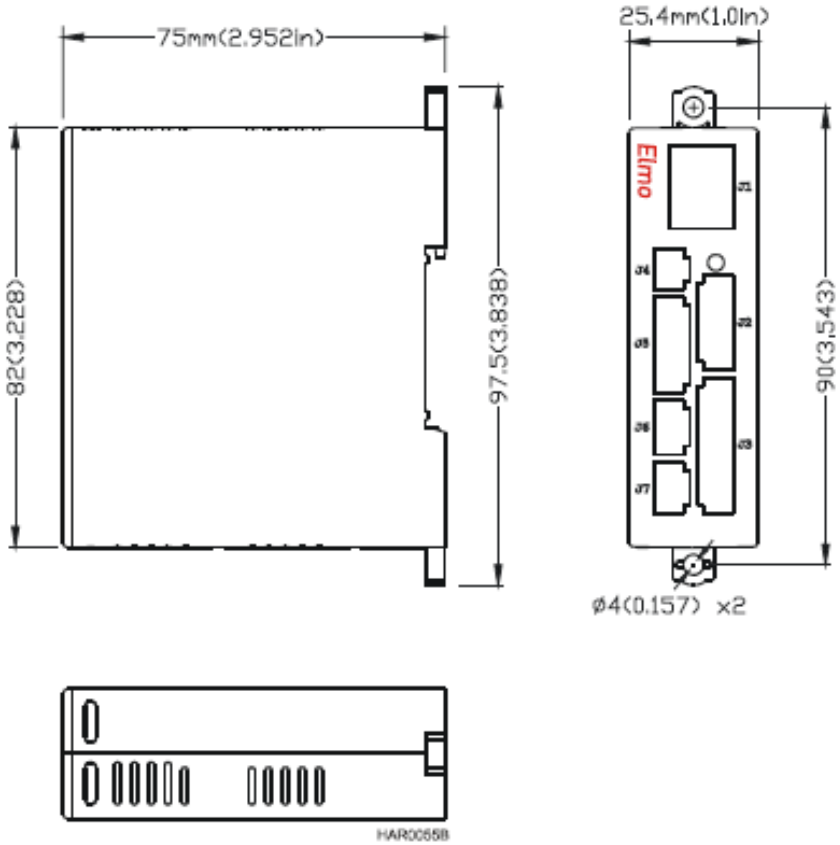
[1] "FS" stands for "Full Scale"

† Stresses beyond those listed under "absolute maximum ratings" may cause permanent damage to the device. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

Specifications can change without prior notification.

Driver Harmonica de Elmo

2. Harmonica Dimensions



3. General Specifications

Feature	Unit	5/60	8/60	12/60	2/100	4/100	8/100	12/100	1/200	2/200	4/200	6/200
Minimum Supply Voltage	VDC	10			20				40			
Nominal Supply Voltage	VDC	55			85				180			
Maximum Supply Voltage	VDC	59			95				195			
Maximum Output Power from the Drive	W	250	410	670	210	320	660	1080	210	330	660	1110
Efficiency at Rated Power	%	> 97										
DC (trapezoidal Commutation) Continuous RMS Current limit (Ic)	A	5	8	13.3	2.5	4	8	13.3	1.25	2	4	6.6
Sinusoidal Commutation Continuous RMS Current limit (Ic)	A	3.5	5.7	9.4	1.8	2.8	5.7	26.6	0.9	1.4	2.8	4.7
Peak current limit (RMS)	A	2 x Ic										
RMS output power without heatsink	%	100	50	20	100	50	20	20	100	50	20	20
PWM Switching Frequency	KHz	22 +/-5% default on the motor										
Switching Method		Advanced Unipolar PWM										
Weight		150 grams (5.3 ounces)										
Dimensions		82 x 25.4 X 75 mm (3.2" x1.0" x 3.0")										
Mounting Method		Wall Mount ("Bookshelf") or DIN Rail										
Digital In / Digital Out / Analog In		6 / 2 / 1										

*50V models are no longer available for new designs

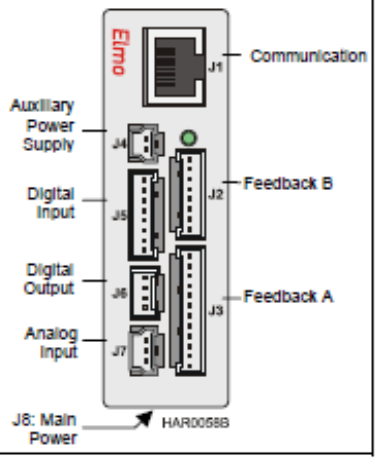
4. Environmental Conditions

Feature	Details
Operating ambient temperature	0° ~ 40° C (32° ~ 104° F)
Storage temperature	-20° ~ +85° C (-4° ~ +185° F)
Humidity	90% maximum non-condensing
Maximum Operating Altitude	10,000m (30,000 feet)
Protection level	IP20

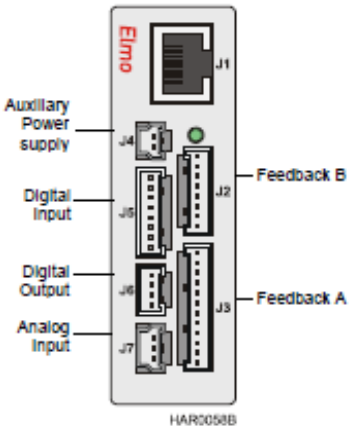
5. Harmonica Connectors

5.1 Connector Types

The table below shows the connector panel of the Harmonica. The *Harmonica Cable Starting Kit* (Cat. No. HAR-CABLEKIT) describes each connector cable in great detail.

Pins	Type	Connector Maker & No. / Mating Plug (on Cable)	Port	Connector Location
8	RJ-45	RJ-45 jack / mates with RJ-45 plug	J1	
8	2 mm Pitch	Molex 35363-0800 / mates with 35507-0800	J2	
12	2 mm Pitch	Molex 35363-1200 mates with 35507-1200	J3	
2	2 mm Pitch	Molex 35363-0200 mates with 35507-0200	J4	
8	2 mm Pitch	Molex 35363-0800 mates with 35507-0800	J5	
4	2 mm Pitch	Molex 35363-0400 mates with 35507-0400	J6	
3	2 mm Pitch	Molex 35363-0300 mates with 35507-0300	J7	
7	5.08 mm Pitch Terminal Block	Phoenix MSTBA 2.5/7-G-5.08 with MSTB 2.5/7-ST-5.08	J8	

5.2 Control and Feedback Connector Specifications

Feature	Details	Connector Location
Product name	Sherlock	
Manufacturer	Molex	
Wire size	24, 26, 28, 30 AWG	
Maximum current	2 A	
Temperature range	-40° to 105° C (-40° to 221° F)	
Plating contact	Tin/Lead (Sn/Pb)	
Maximum voltage	125 V	
Contact resistance	< 20 mΩ	
Withstanding voltage	500 VAC	
Insulation resistance	> 1000 MΩ	
Terminal contact	Phosphor bronze	
UL files	E29179, UL 94 V-0	
Cable connector	Molex 35507-XX00, where XX is the number of leads	
Hand crimper	Molex 63811-1200	
Crimp terminal	Molex 50212	